

# Ratel - Solid Mechanics for the Exascale Era

Jeremy L Thompson

University of Colorado Boulder

*jeremy@jeremylt.org*

Website/CV: <https://jeremylt.org>

6 August 2025

# Overview

1 libCEED

2 RateL

3 GPU and AD

4 Contact and MPM

5 Questions

# Top 500

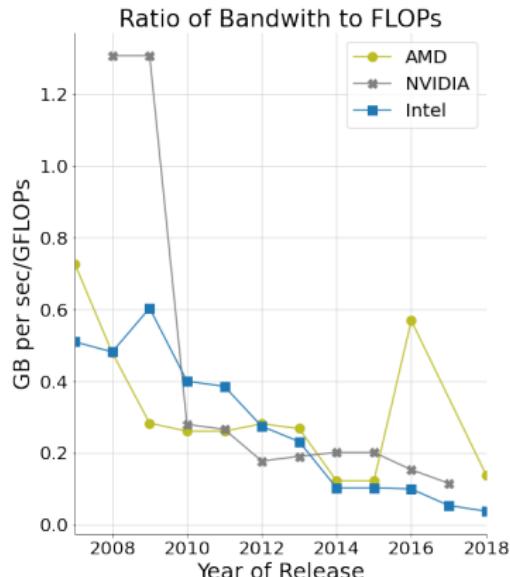
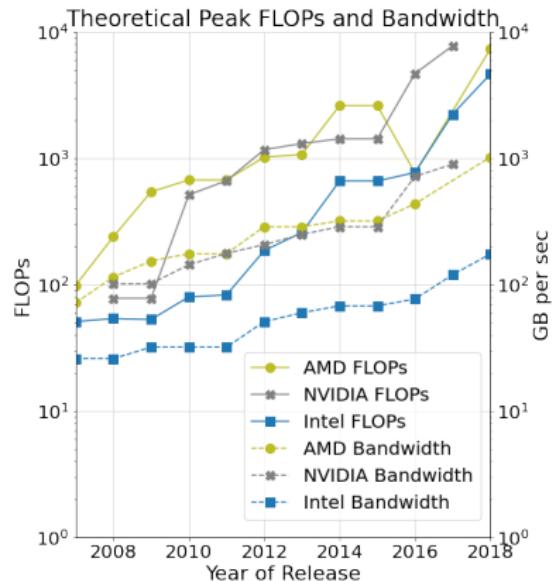
| Machine    | HPL             | HPCG         |
|------------|-----------------|--------------|
| El Capitan | 1,742.00 PFLOPs | 17.41 PFLOPS |
| Fugaku     | 442.01 PFLOPs   | 16.00 PFLOPS |
| Frontier   | 1,353.00 PFLOPs | 14.05 PFLOPS |
| Aurora     | 1,012.00 PFLOPs | 5.61 PFLOPS  |
| LUMI       | 379.70 PFLOPs   | 4.59 PFLOPS  |

Top 500 Machines for HPCG with HPL peak FLOPs

HPCG closer to representative FLOPs for simulations

Difficult to realize peak FLOPs with CG on modern machines

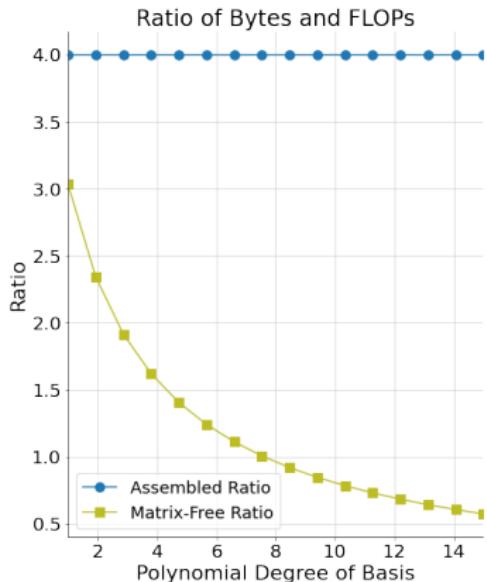
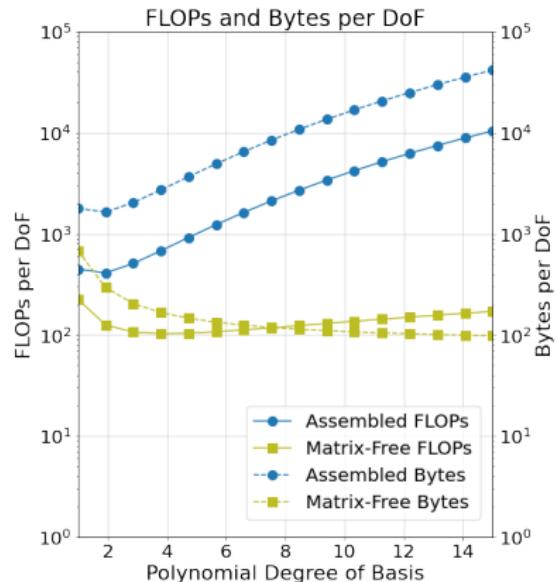
# Modern Hardware



Memory bandwidth is improving slower than FLOPs

Mirrors difference between Top 500 HPL vs HPCG benchmarks

# Benefits of Matrix-Free



Requirements for matrix-vector product with sparse matrix vs matrix-free  
for screened Poisson  $\nabla^2 u - \alpha^2 u = f$  in 3D

**Matrix-free representations using tensor product bases  
better match modern hardware**

# libCEED Team

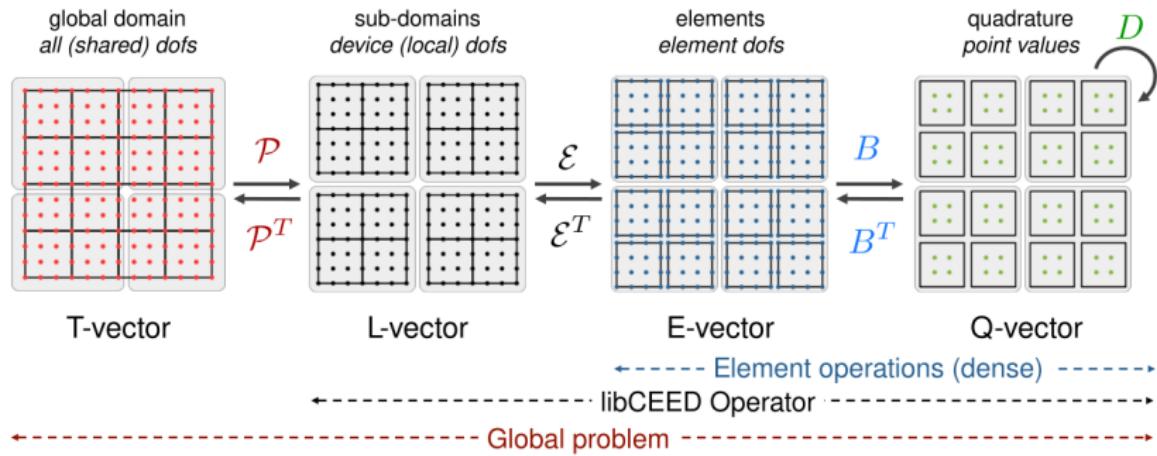


libCEED Repo: <https://github.com/CEED/libCEED>

Developers: Ahmad Abdelfattah, Zach R. Atkins, Valeria Barra,  
Natalie Beams, Jed Brown, Jean-Sylvain Camier,  
Veselin Dobrev, Yohann Dudouit, Leila Ghaffari,  
Sebastian Grimberg, Tzanio Kolev, David Medina,  
Will Paznel, Thilina Ratnayaka, Rezgar Shakeri,  
Stan Tomov, James Wright III, Jeremy L Thompson

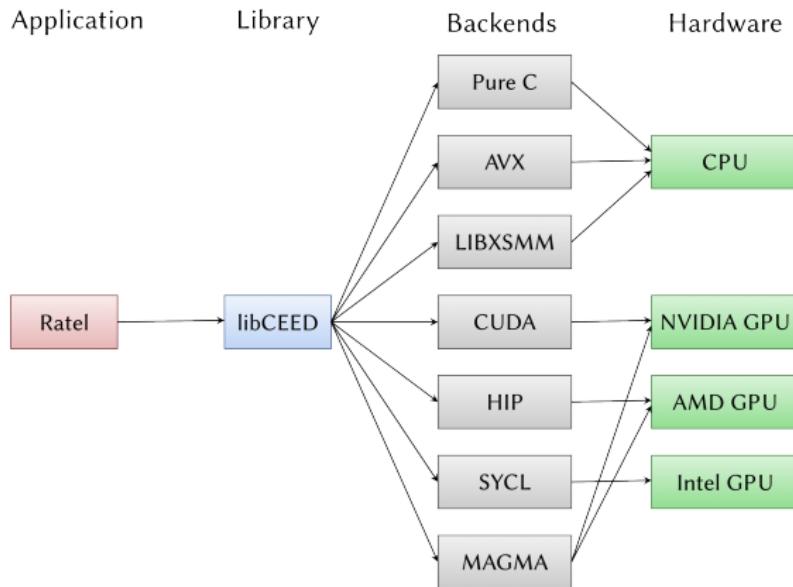
# Matrix-Free Operators with libCEED

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



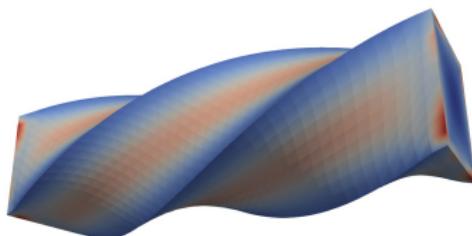
libCEED provides arbitrary order matrix-free operator evaluation

# Performance Portability from libCEED

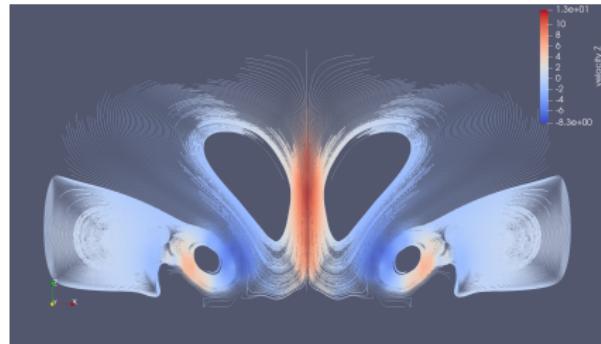


Performance portability with libCEED's matrix-free operators

# Mini-Apps



- 7.9e-02  
- 0.07  
- 0.06  
- 0.05  
- 0.04  
- 0.03  
- 0.02  
- 0.01  
- 1.0e-04



libCEED solid mechanics (left) and fluid dynamics (right) mini-apps

- libCEED supports FEM-like simulations on modern hardware
- Mini-apps have been expanded into independent libraries

# libCEED Projects

Several projects built using libCEED

- RateL - solid mechanics FEM and iMPM (PSAAP)
- HONEE - fluid dynamics FEM & differential filtering (PHASTA)
- MFEM - various applications, libCEED integrators (LLNL)
- Palace - quantum circuit design, MFEM + libCEED (Amazon)
- RDycore - FV river dynamical core, PETSc + libCEED (SciDAC)

# Design Implications

Using matrix-free operators drives design decisions

- Direct solvers are out (assembled matrices,  $\mathcal{O}(p^6)$ )
- Iterative solvers are in (Krylov methods, etc)
- High(er) order = high accuracy & bad condition numbers
- Preconditioning is needed for fast convergence

# Ratels Team

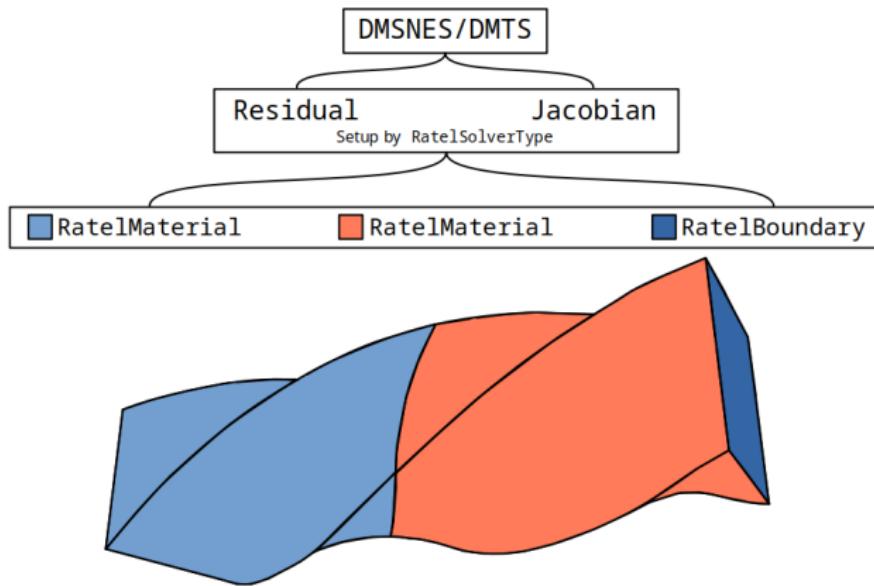


Ratels Repo: <https://gitlab.com/micromorph/ratels>

Developers: Zach R. Atkins, Jed Brown, Fabio Di Gioacchino,  
Layla Ghaffari, Zach Irwin, Rezgar Shakeri,  
Ren Stengel, Jeremy L Thompson

PSAAP Micromorph Center & iMPM

# Basic Design



Each material region sets up part of the non-linear and linear equations

# Material Models

Multiple models are supported in RateL

- Elasticity - Linear, Neo-Hookean, Mooney Rivlin, Ogden, Hencky
- Mixed Elasticity - Linear, Neo-Hookean, Odgen
- Plasticity - Hencky, various types
- Anisotropy - Neo-Hookean
- Brittle Fracture - Linear damage, Neo Hookean damage
- Poromechanics - Linear, Neo-Hookean
- Viscoelasticity - Hencky
- iMPM - Neo-Hookean, Neo-Hookean damage

Current and initial configuration for many models  
(Rezgar Shakeri, Fabio Di Gioacchino, Zach Irwin, PSAAP postdocs)

# Boundary Conditions

Multiple boundary conditions are supported in RateL

- Clamp (Dirichlet)
- Slip (partial Dirichlet)
- Traction (Neumann)
- Pressure
- Rigid contact - Nitsche or penalty, with Coulomb or Threlfall friction

All BCs are time dependent

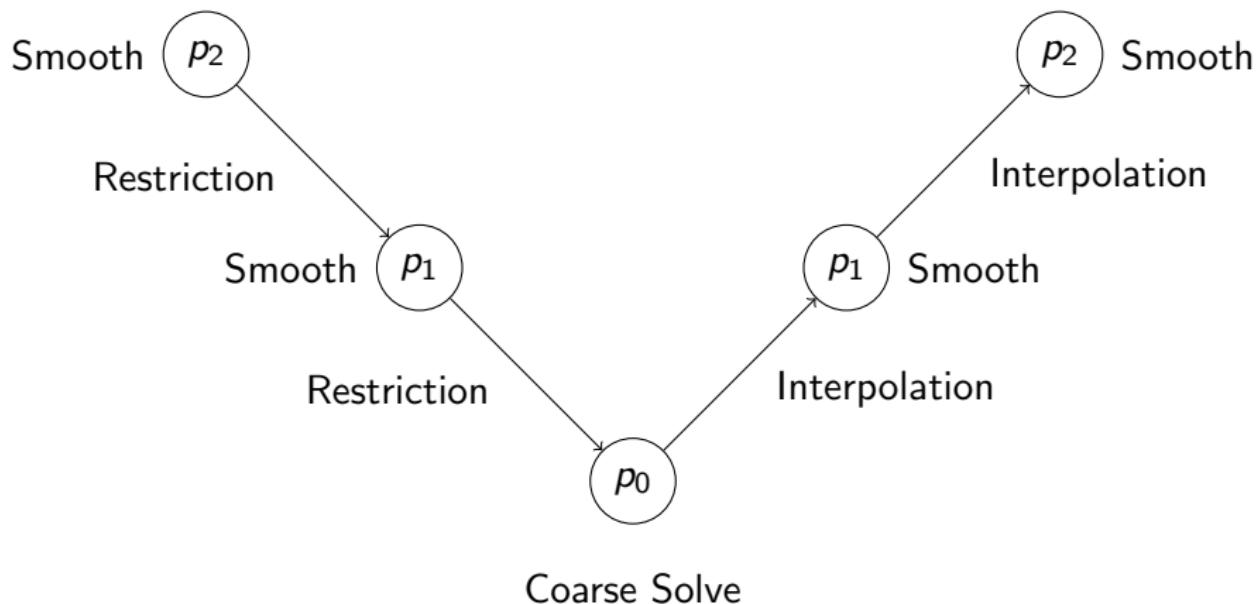
(Zach R. Atkins, PhD student and Rezgar Shakeri, PSAAP postdoc)

# Preconditioning Support

Iterative solvers need preconditioning

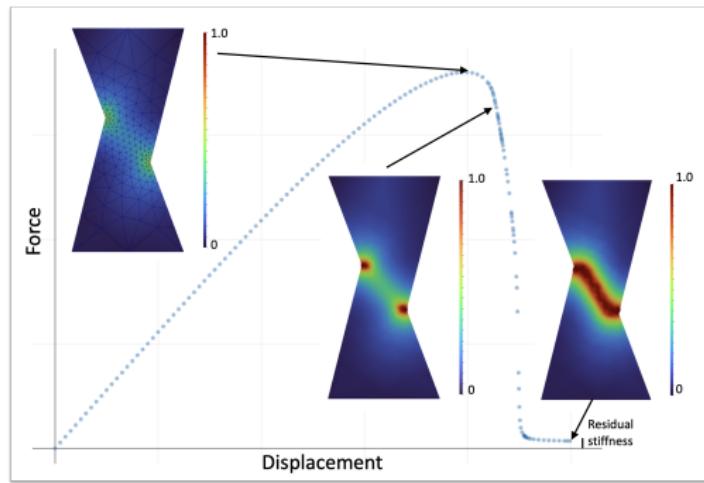
- $Ax = b$ , slow for high-order (ill-conditioned)
- libCEED supports various preconditioner ingredients
- Most PETSc preconditioners fully supported
- Multigrid prolong/restrict also supported in libCEED

# p-multigrid



RateL uses matrix-free p-multigrid, AMG coarse solve

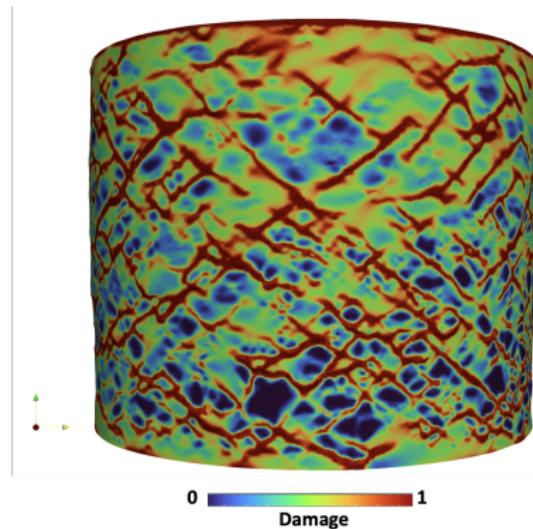
# Example - Linear Damage



```
$ bin/ratel-quasistatic -options_file examples/ymls/ex02-  
quasistatic-elasticity-linear-damage-compressiveshear-  
AT2-face-forces.yml
```

Quasistatic simulation of compressive shear for generic brittle material  
(Fabio Di Gioacchino, PSAAP postdoc)

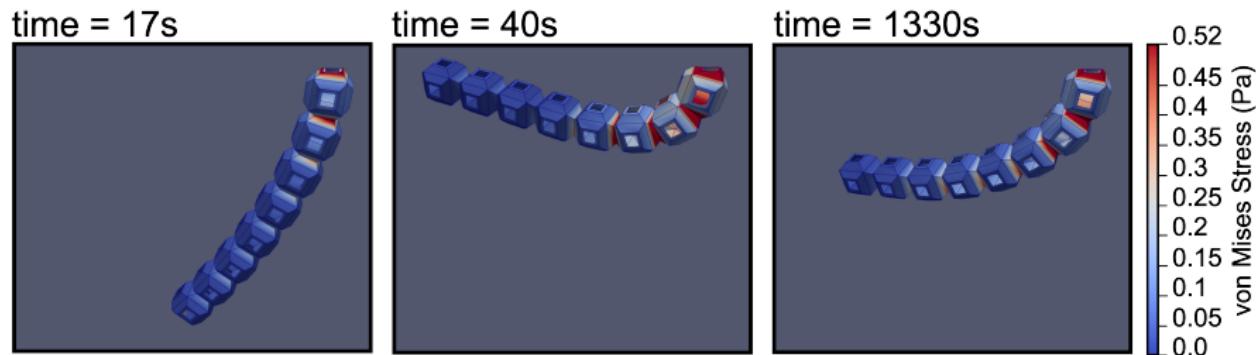
# Example - Hencky Viscoelasticity Damage



Quasistatic simulation of IDOX grains in IDOX/Estane matrix  
with damage due to unconfined compression

(Fabio Di Gioacchino, PSAAP postdoc)

# Example - Dynamic Pendulum



```
$ bin/ratel-dynamic -options_file examples/ymls/ex03-dynamic  
-elasticity-schwarz-pendulum-enzyme.yml
```

Dynamic simulation of Neo-Hookean Schwarz-P 'pendulum' with Enzyme  
(Layla Ghaffari, recent PhD graduate)

# CEED Benchmark Problems

## Performance on CEED BPs

- BP1 - Scalar projection problem
- BP2 - 3 component projection problem
- BP3 - Scalar Poisson problem
- BP4 - 3 component Poisson problem
- Ogden - representative production problem

Bulk of FLOPs are in basis evaluation for BPs

# CEED Benchmark Problems

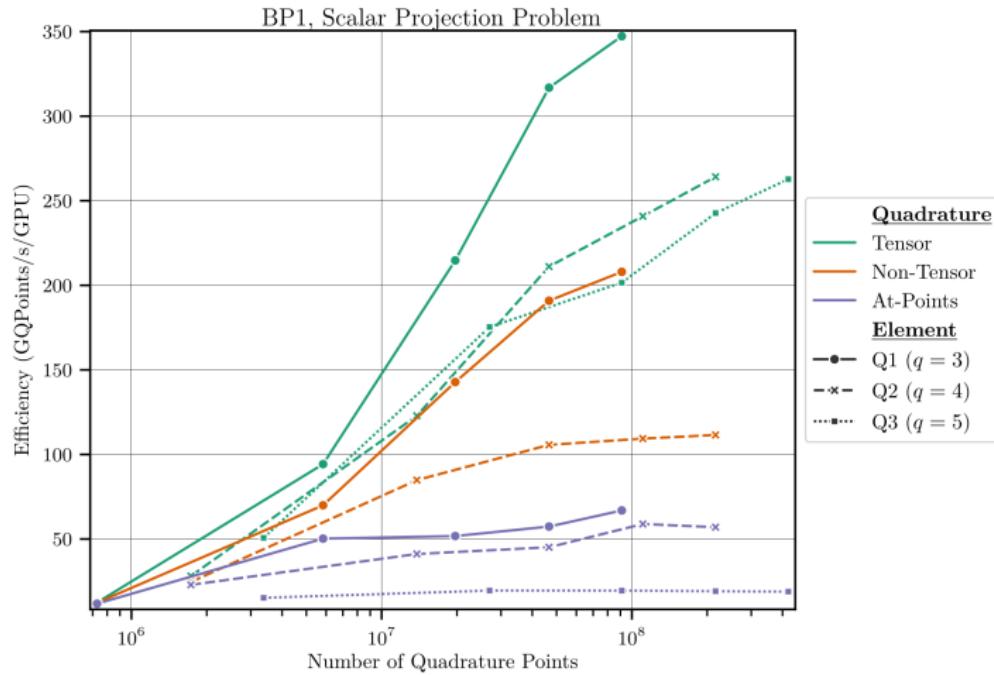
## Performance on CEED BPs

- $p = 2, 3, 4$  and  $q = p + 1$
- Units cube with  $30^3$ ,  $60^3$ ,  $90^3$ ,  $120^3$ , and  $150^3$  elements
- Compare tensor, non-tensor, and at-points basis evaluation
- MMS w/ partial sum of Weierstrass function,  $a = 0.5$ ,  $b = 1.5$ ,  $N = 2$

Using 4x AMD Instinct™MI300A Accelerated Processing Units (APUs)

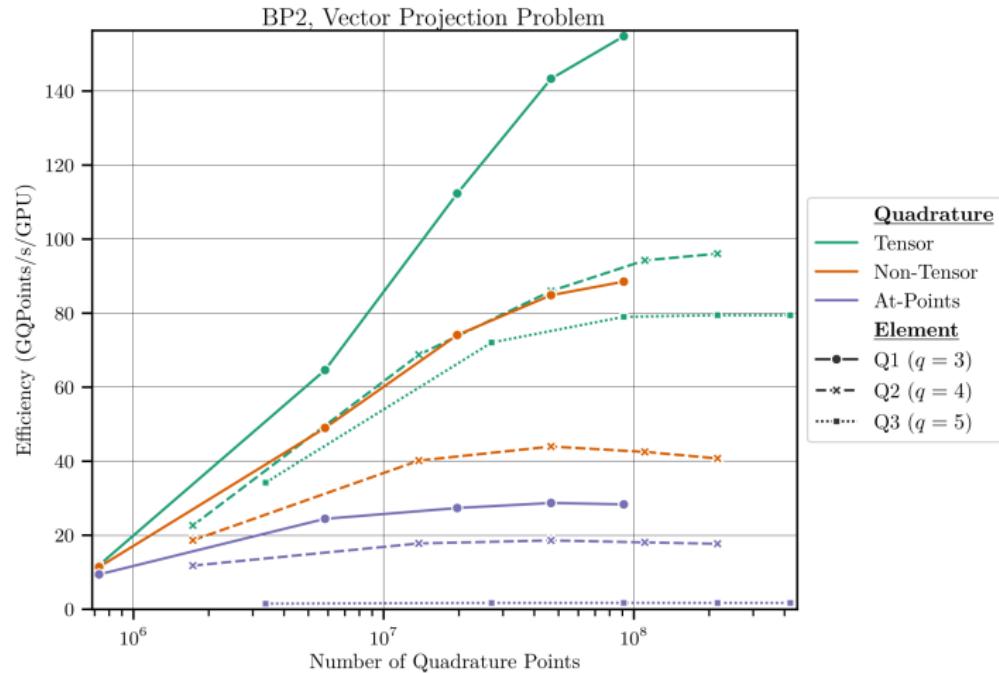
See also Performance Portable Solid Mechanics via Matrix-Free p-Multigrid  
(Also, dissertation of Ren Stengel, recent PhD graduate)

## BP1



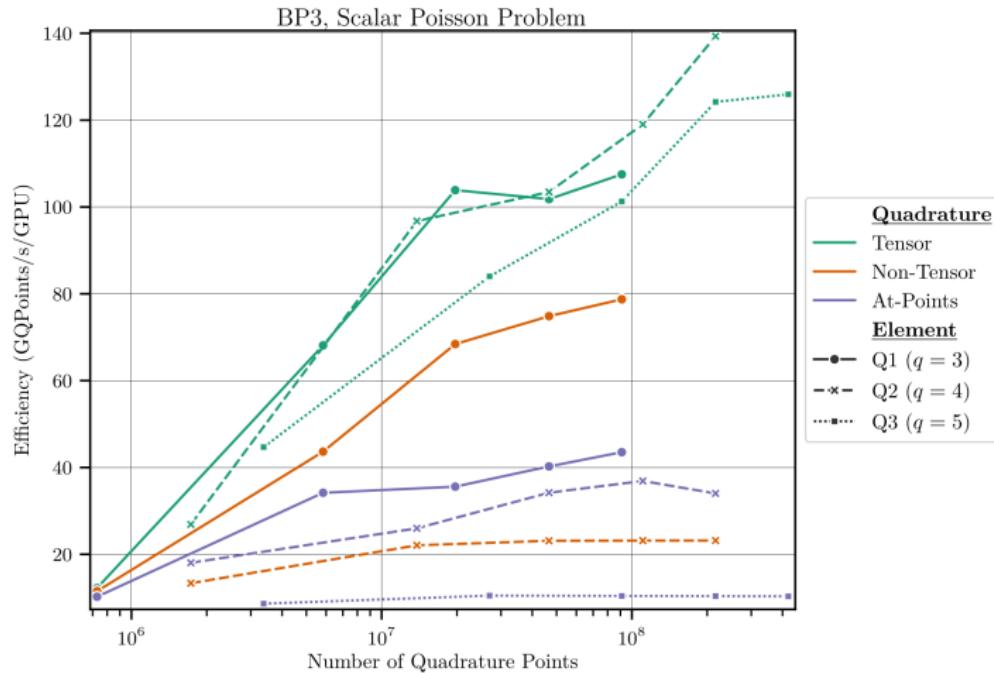
More FLOPs to do leads to lower efficiency

## BP2



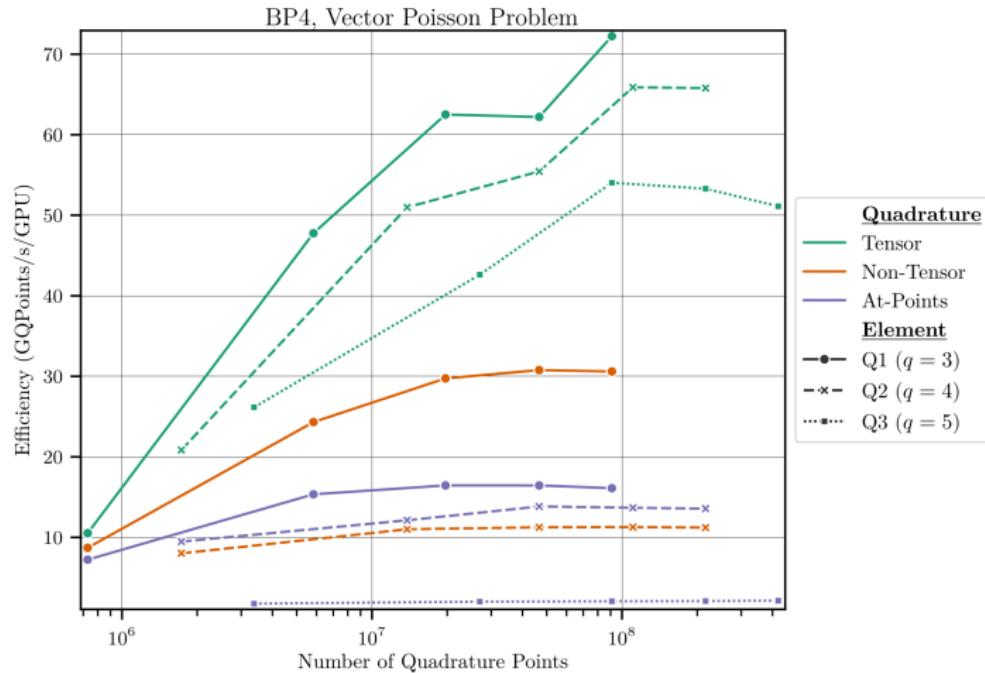
More components better represents production workloads

## BP3



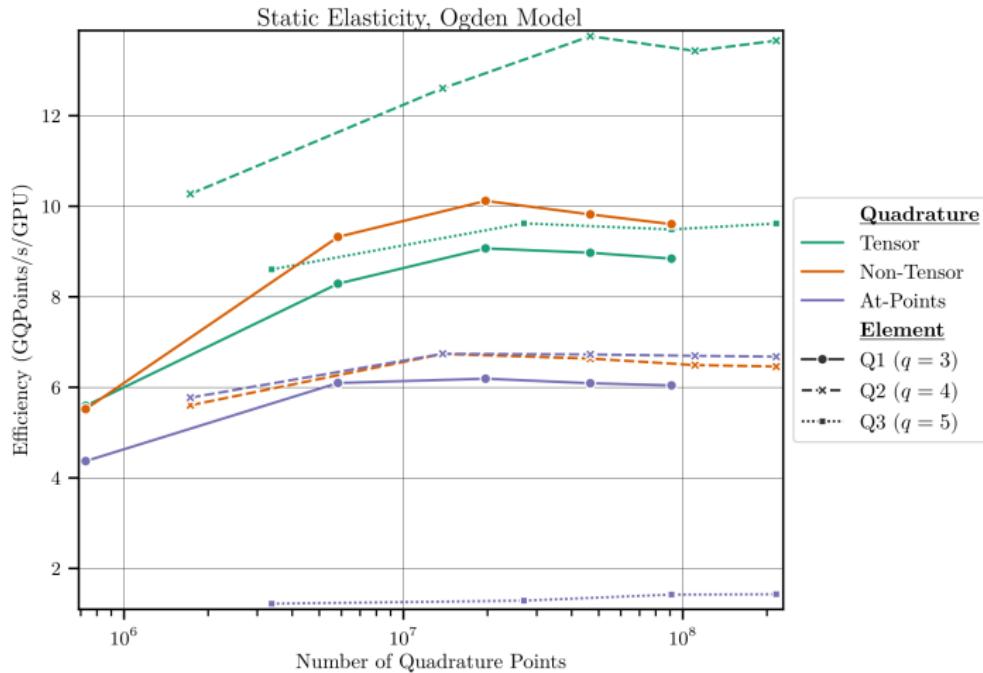
With derivatives, at-points closer to non-tensor

## BP4



Closest CEED BP to production workload

# Ogden



Basis cost less important with heavier QFunctions

# Two Families of Approaches

Three libCEED backends with two approaches to operator application

- Separate kernels
  - `/gpu/*/ref` and `/gpu/*/shared`
  - $\mathcal{E}$ ,  $\mathcal{B}$ , and  $\mathcal{D}$  all separate kernels
  - Higher overall memory usage, multiple kernel launches
- Fused kernel
  - `/gpu/*/gen`
  - Single kernel JiTed with data from  $\mathcal{E}$ ,  $\mathcal{B}$ , and  $\mathcal{D}$
  - Lower overall memory usage, single kernel launch

# Ref Operator Application

`/gpu/*/ref` and `/gpu/*/shared` use largely the same code

$A_L = \mathcal{E}^T B^T D B \mathcal{E}$  use separate kernels

$\mathcal{E}$  source comes from the `/gpu/*/ref`

`/gpu/*/ref` uses basic kernels for  $B$

`/gpu/*/shared` uses shared memory for  $B$

$D$  source is given by the user

# Gen Operator Application

`/gpu/*/gen` generates a single kernel for the operator

$A_L = \mathcal{E}^T B^T D B \mathcal{E}$  uses a single kernel

$\mathcal{E}$  source comes from the `/gpu/*/ref`

$B$  source comes from `/gpu/*/shared`

$D$  source is given by the user

(Original gen backend by Yohann Dudouit)

# Generated Operator Kernel

```
1 extern "C" __global__ void CeedKernelCudaGenOperator_mass(CeedInt num_elem,
2     void* ctx, FieldsInt_Cuda indices, Fields_Cuda fields, Fields_Cuda B,
3     Fields_Cuda G, CeedScalar *W, Points_Cuda points) {
4     // Setup kernel data
5
6     // Input and Output field constants and basis data
7
8     // Element loop
9     __syncthreads();
10    for (CeedInt elem = blockIdx.x*blockDim.z + threadIdx.z; elem < num_elem;
11        elem += blockDim.x*blockDim.z) {
12        // -- Input field restrictions (E) and basis actions (B)
13
14        // -- Output field setup
15        {
16            // -- Apply QFunction (D)
17            mass(ctx, 1, inputs, outputs);
18        }
19
20        // -- Output field basis actions (B^T) and restrictions (E^T)
21    }
22 }
23 // -----
```

$$\mathbf{A}_L = \mathcal{E}^T \mathbf{B}^T \mathbf{D} \mathbf{B} \mathcal{E} \text{ in a single kernel}$$

# QFunctions in Rust



- libCEED can compile *D* in Rust for CUDA
- Lower Rust and `/gpu/cuda/gen` kernel to LLVM IR
- Link, optimize, and compile LLVM IR to PTX, similar final perf
- <https://github.com/CEED/libCEED/pull/1881>

(Summer undergrad Allen MacFarland)

# UHYPER Integration

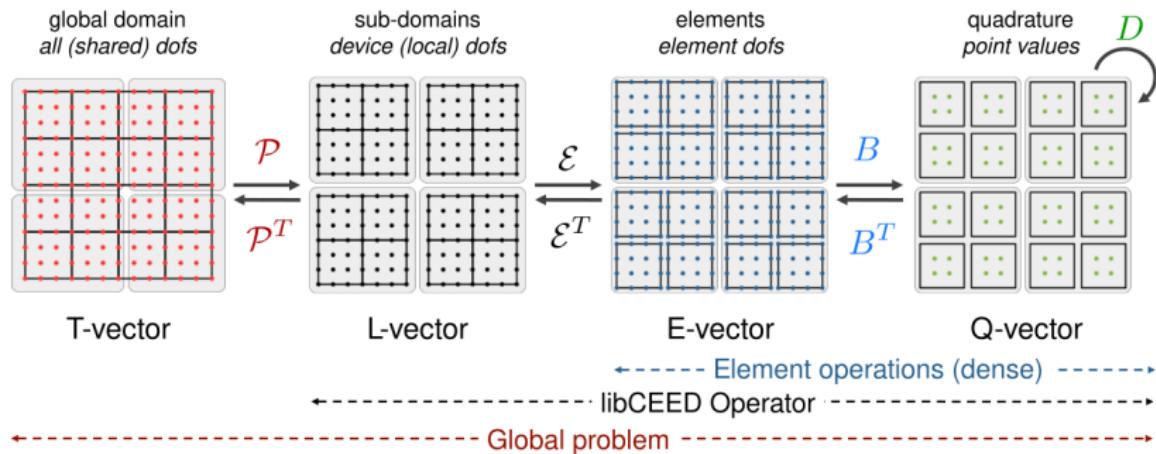


- Ongoing work to wrap UHyper for Ratel
- CPU implementation working with Flang
- Hope to use LLVM IR for future CUDA support
- [https://gitlab.com/micromorph/ratel/-/merge\\_requests/1136](https://gitlab.com/micromorph/ratel/-/merge_requests/1136)

(Summer undergrad Adonay Mezgebe)

# AD Roadmap

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



AD can be implemented in  $D$

# Enzyme AD



- Computes gradients of source code
- Uses optimized LLVM IR
- Performance similar to hand derivatives w/o algebraic simplification\*

(See dissertation of Layla Ghaffari, recent PhD graduate)

# Enzyme Neo-Hookean Elasticity

```

1 CEED_QFUNCTION_HELPER void RateKirchhofftau_sym_NeoHookean_AD(const CeedScalar
2     lambda, const CeedScalar mu, CeedScalar e_sym[6], CeedScalar tau_sym[6]) {
3
4     CeedScalar dPsi_sym[6] = {0.}, b_sym[6], dPsi[3][3], b[3][3], tau[3][3];
5
6     // dPsi / de
7     __enzyme_autodiff((void *)RateStrainEnergy_NeoHookeanCurrentAD_Enzyme, e_sym,
8         dPsi_sym, enzyme_const, lambda, enzyme_const, mu);
9     for (CeedInt i = 3; i < 6; i++) dPsi_sym[i] /= 2.;
10
11    // b = 2 e + I
12    for (CeedInt j = 0; j < 6; j++) b_sym[j] = 2 * e_sym[j] + (j < 3);
13
14    // tau = (dPsi / de) b
15    RatelSymmetricMatUnpack(dPsi_sym, dPsi);
16    RatelSymmetricMatUnpack(b_sym, b);
17    RatelMatMatMult(1., dPsi, b, tau);
18    RatelSymmetricMatPack(tau, tau_sym);
19
20 }
21
22 CEED_QFUNCTION_HELPER void Rateltau_fwd(const CeedScalar lambda, const CeedScalar mu
23     , CeedScalar e_sym[6], CeedScalar de_sym[6], CeedScalar tau_sym[6], CeedScalar
24     dtau_sym[6]) {
25     __enzyme_fwddiff((void *)RateKirchhofftau_sym_NeoHookean_AD, enzyme_const, lambda,
26         enzyme_const, mu, e_sym, de_sym, tau_sym, dtau_sym);
27 }
```

Enzyme computing Jacobian from Residual, via energy

# Better Enzyme AD



- Enzyme AD available in Rust
- Simplifies user experience, route to easier GPU impl
- Goal of similar perf to hand derivatives of  $D$  in C
- <https://github.com/EnzymeAD/rust>

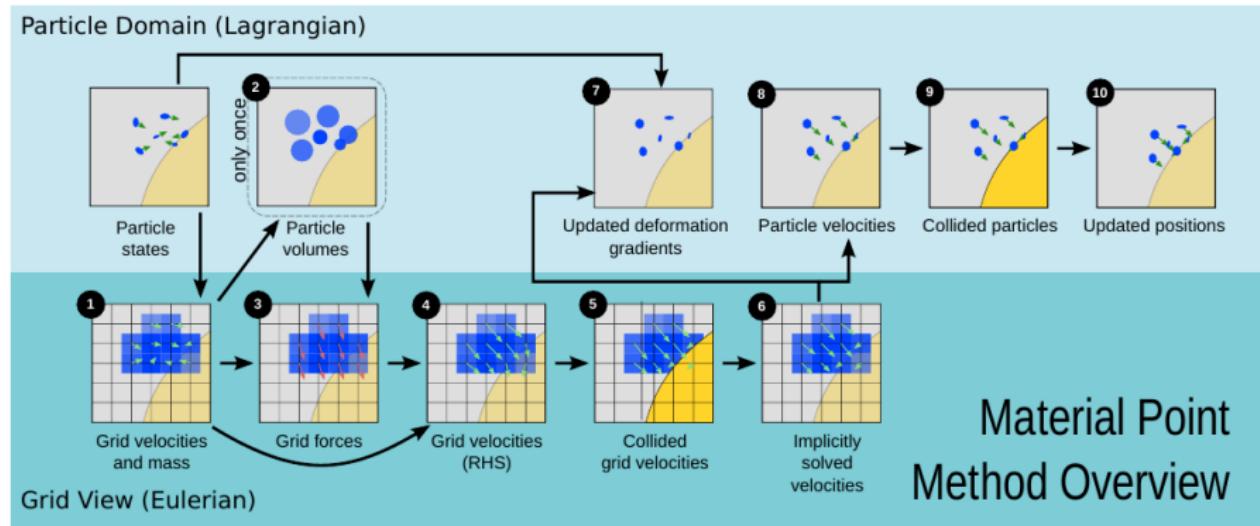
# Contact Models

## Nitsche method with rigid platens

- Support both Nitsche and penalty method
- Contact shape either platen or cylinder
- Support time dependent translation of surface
- Coulomb or Threlfall friction
- Deformable mesh-to-mesh contact in progress

(Zach R. Atkins, PhD student)

# What is MPM?



- Continuum based particle method with background mesh for gradients
- Extension of FLIP (which is an extension of PIC)
- Used in rendering for the movie *Frozen*

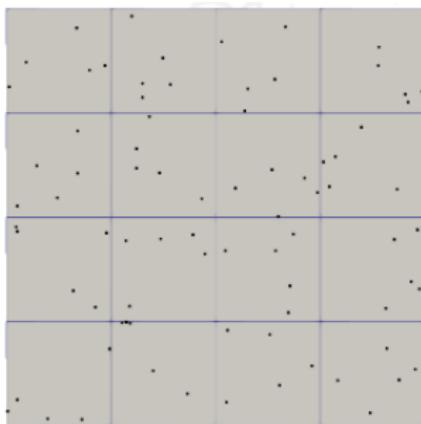
# MPM vs FEM

MPM can be formulated as very similar to FEM

- Problem on background mesh changes when material points move
- Can be viewed as FEM with arbitrary quadrature point locations
- Natural fit for libCEED matrix-free representation
- RateL FEM infrastructure provides fast background mesh solves

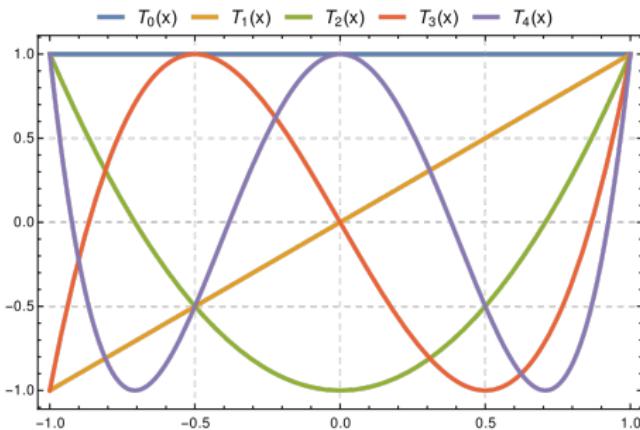
(Zach R. Atkins, PhD student)

# DMSwarm for Material Points



- PETSc DMSwarm manages material points
- Point migration on CPU only (Toulumne/EI Capitan helps here!)
- PETSc DMFlex manages background mesh

# libCEED Basis Evaluation to Points



- Interpolate from primal to dual (quadrature) space
- Fit Chebyshev polynomials to values at quadrature points
- Evaluate Chebyshev polynomials at arbitrary points

# libCEED Basis Evaluation to Points

Interpolation to Chebyshev has same FLOPs as FEM  $\mathcal{O}(q^4)$

- Invert map  $C^{-1}$  from quadrature points to Chebyshev coeffs
- Create 1D interpolation matrix  $B = CN$
- Tensor product:  
$$B = (C \otimes C \otimes C) (N \otimes N \otimes N) = (CN) \otimes (CN) \otimes (CN)$$
- Additional cost from evaluation to arbitrary points

# libCEED Basis Evaluation to Points

Per point evaluation is expensive  $\mathcal{O}(q^3)$

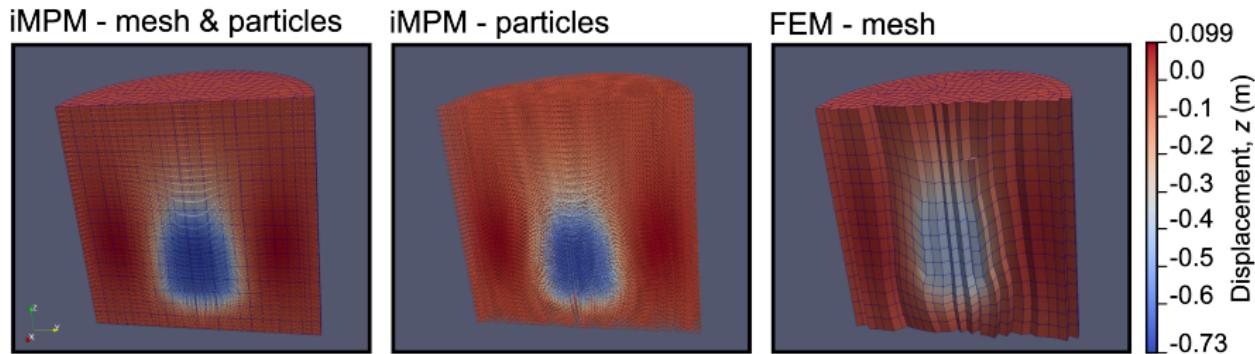
- Recurrence for Chebyshev values at point

$$f_0 = 1, f_1 = 2x, f_n = 2xf_{n-1} - f_{n-2}$$

$$f'_0 = 0, f'_1 = 2, f'_n = 2xf'_{n-1} + 2f_{n-1} - f'_{n-2}$$

- Contract pencil of values with element coefficients
- Evaluation is independent per material point
- $\mathcal{O}(q^3)$  FLOPs at  $\mathcal{O}(\hat{q}^3)$  points
- Using  $p = q, \hat{q} = q + 1$  in current work

# Example - MPM Sinker

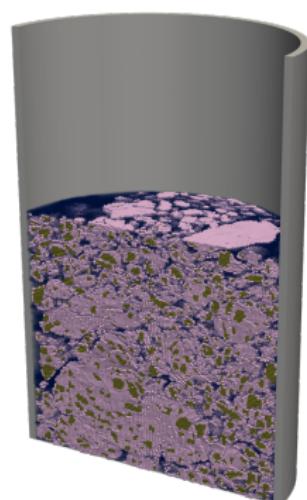
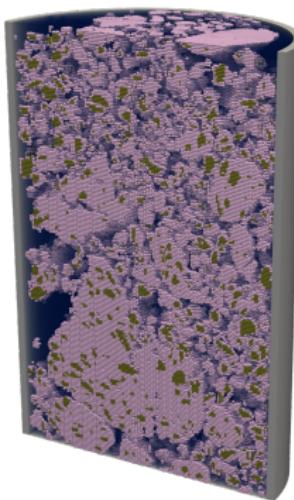
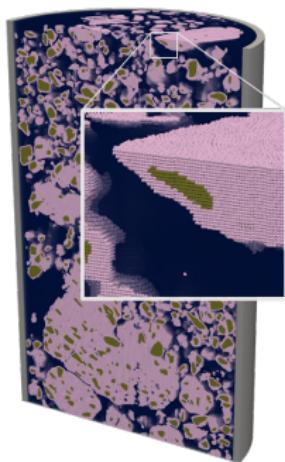


```
$ bin/ratet-quasistatic -options_file examples/ymls/ex02-  
quasistatic-elasticity-mpm-neo-hookean-damage-current-  
sinker-cylinder.yml
```

FEM, iMPM simulations of dense sinker in "foam" validation problem  
(Mesh distortion limits FEM simulation)

# Example - Press Simulation

316 MPoints



Compression of mock HE grains (gold) and binder (pink) mixture  
(Reset background mesh to computational region on each timestep)

# Ratel PCpMG

pMG giving promising initial results with GPU impl

- Neo-Hookean finite strain elasticity with damage
- Confined press of grain/binder with "sticky air" voids
- Jacobi iterations tend to double with 2x refinement
- pMG iteration counts robust with refinement

|        | Material Points | Jacobi its | pMG its |
|--------|-----------------|------------|---------|
| Coarse | 388,800         | 900-1000   | 35-45   |
| Fine   | 7,372,800       | -          | 25-40   |

# Questions?



libCEED Repo: <https://github.com/CEED/libCEED>

Ratel Repo: <https://gitlab.com/micromorph/ratel>

Grant: Predictive Science Academic Alliance Program (DE-NA0003962)



University of Colorado  
Boulder



# Ratel - Solid Mechanics for the Exascale Era

Jeremy L Thompson

University of Colorado Boulder

*jeremy@jeremylt.org*

Website/CV: <https://jeremylt.org>

6 August 2025