Pragmatic performance-portable solids and fluids with Ratel, libCEED, and PETSc

Jed Brown, CU Boulder

Collaborators: Zach Atkins, Valeria Barra, Natalie Beams, Fabio Di Gioacchino, Leila Ghaffari, Ken Jansen, Matthew Knepley, William Moses, Rezgar Shakeri, Karen Stengel, Jeremy L. Thompson, James Wright III, Junchao Zhang

NUWEST 2024

David Keyes, "Petaflop/s, seriously" (ca. 2007)

Gedanken experiment:

How to use a jar of peanut butter as its price slides downward?

- In 2007, at $3.20: make sandwiches
- By 2010, at $0.80: make recipe substitutions for other oils
- By 2013, at $0.20: use as feedstock for biopolymers, plastics, etc.
- By 2016, at $0.05: heat homes
- By 2019, at $0.0125: pave roads ☺

The cost of computing has been on a curve *much better than this* for two decades and promises to continue for at least one more. Like everyone else, scientists should plan increasing uses for it…

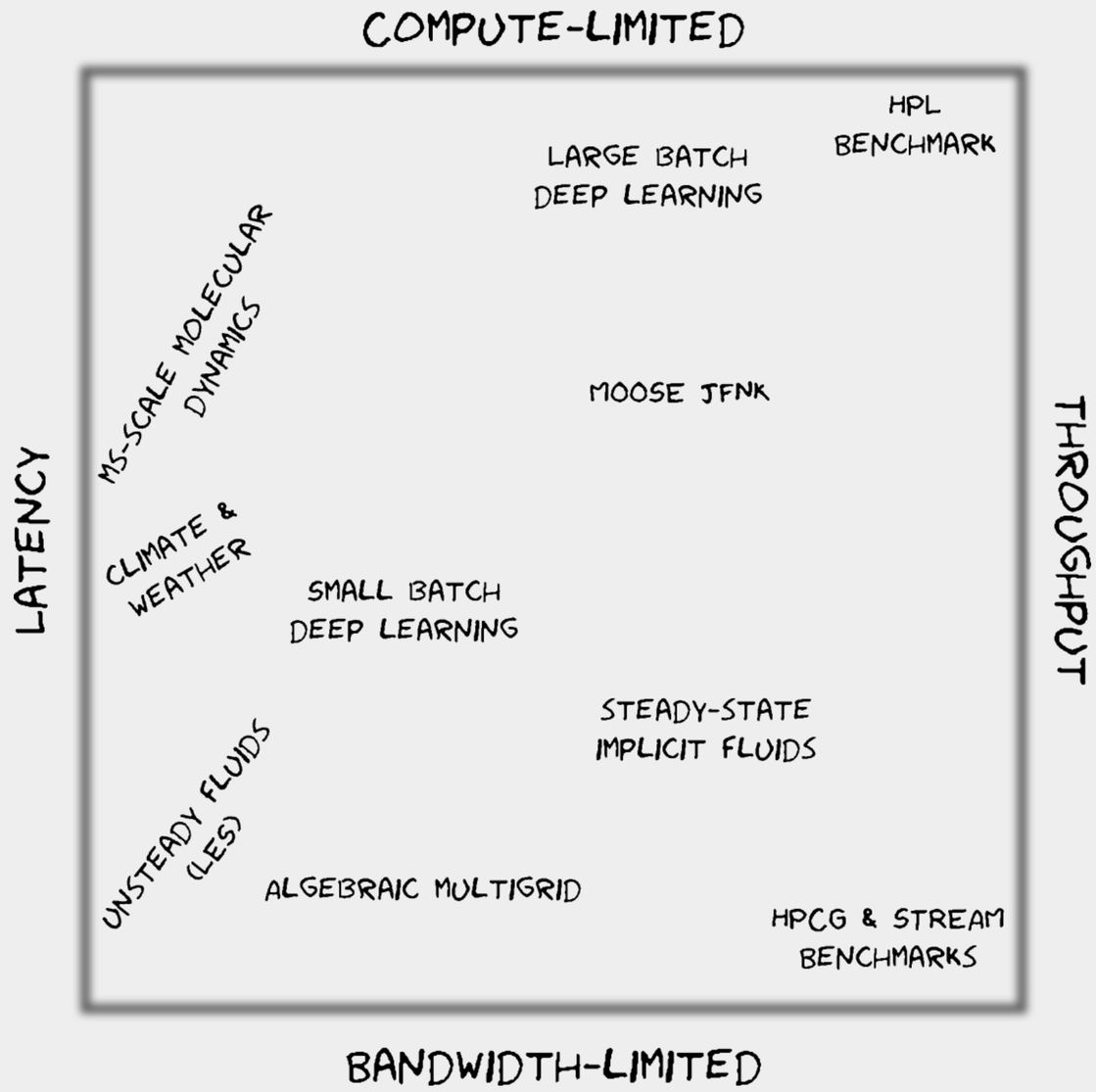
Constants matter

Relative cost of compute versus memory access

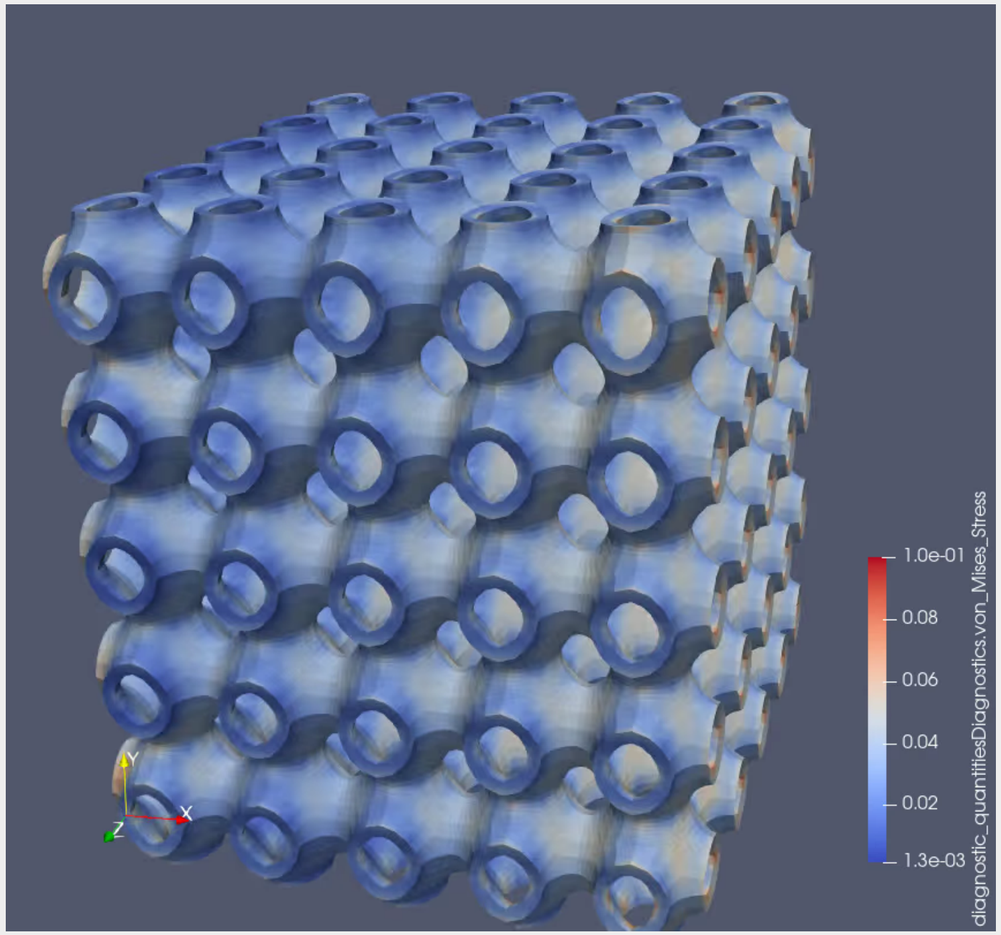
Accuracy tolerances depend on application

GPU vs CPU latencies

Accuracy or conservation?
Unbiased or biased error?



Nonlinear solid mechanics

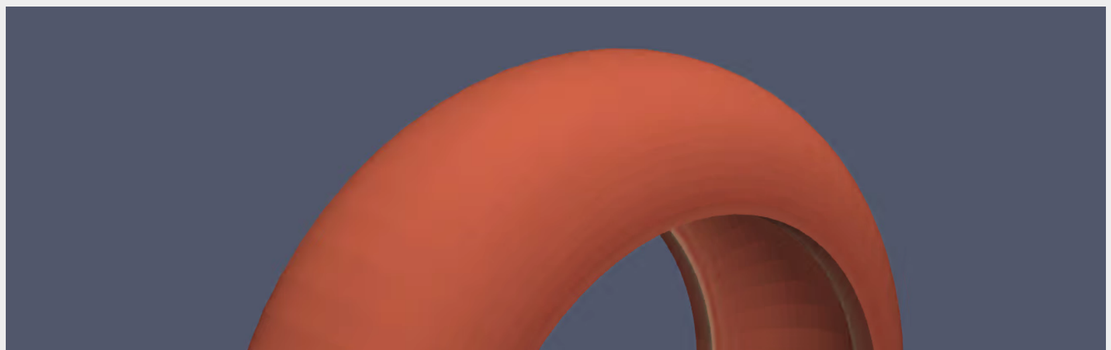


Industrial state of practice

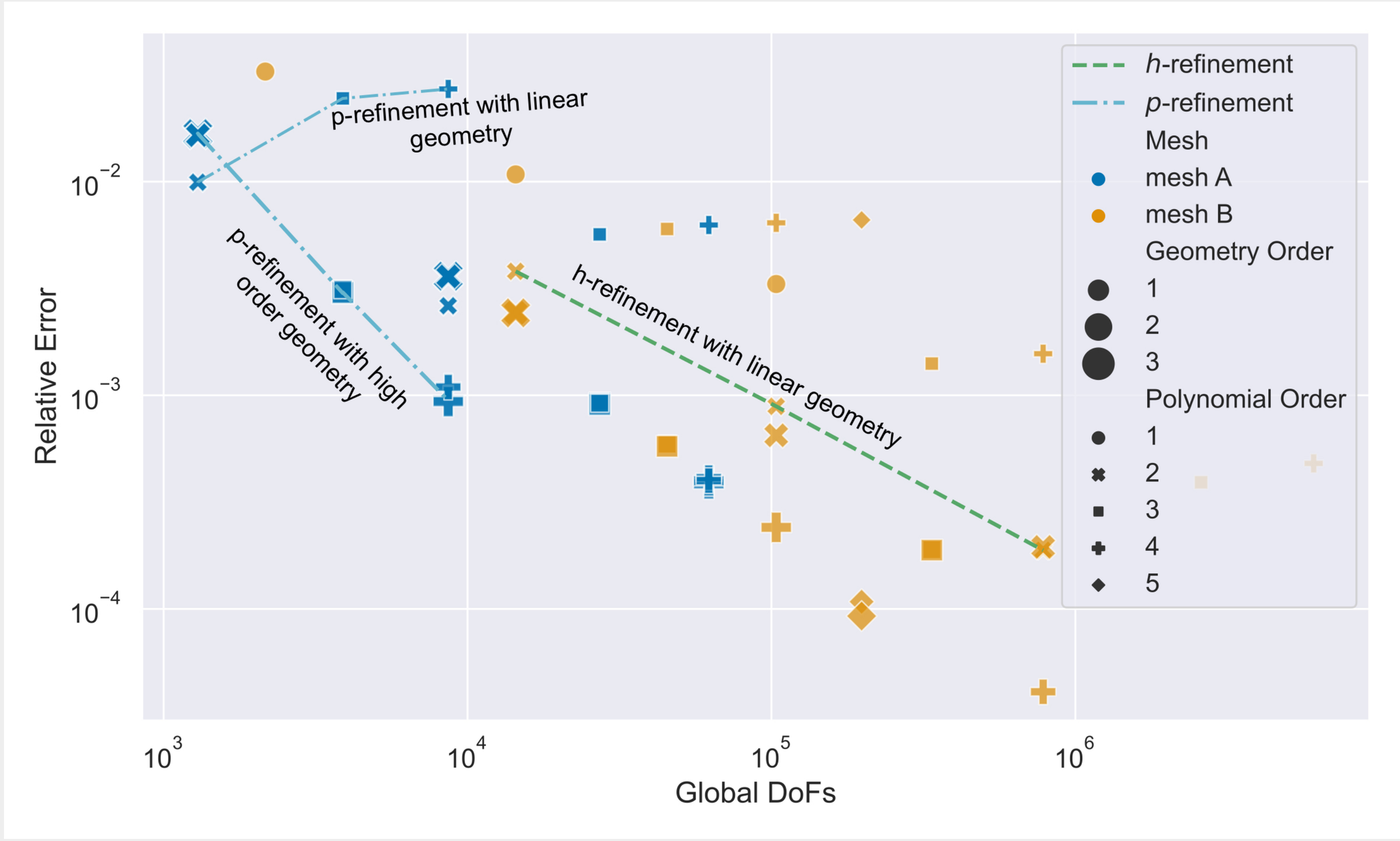
- Low order finite elements: Q_1 (trilinear) hexahedra, P_2 (quadratic) tetrahedra.
- Assembled matrices, sparse direct and algebraic multigrid solvers

Myths

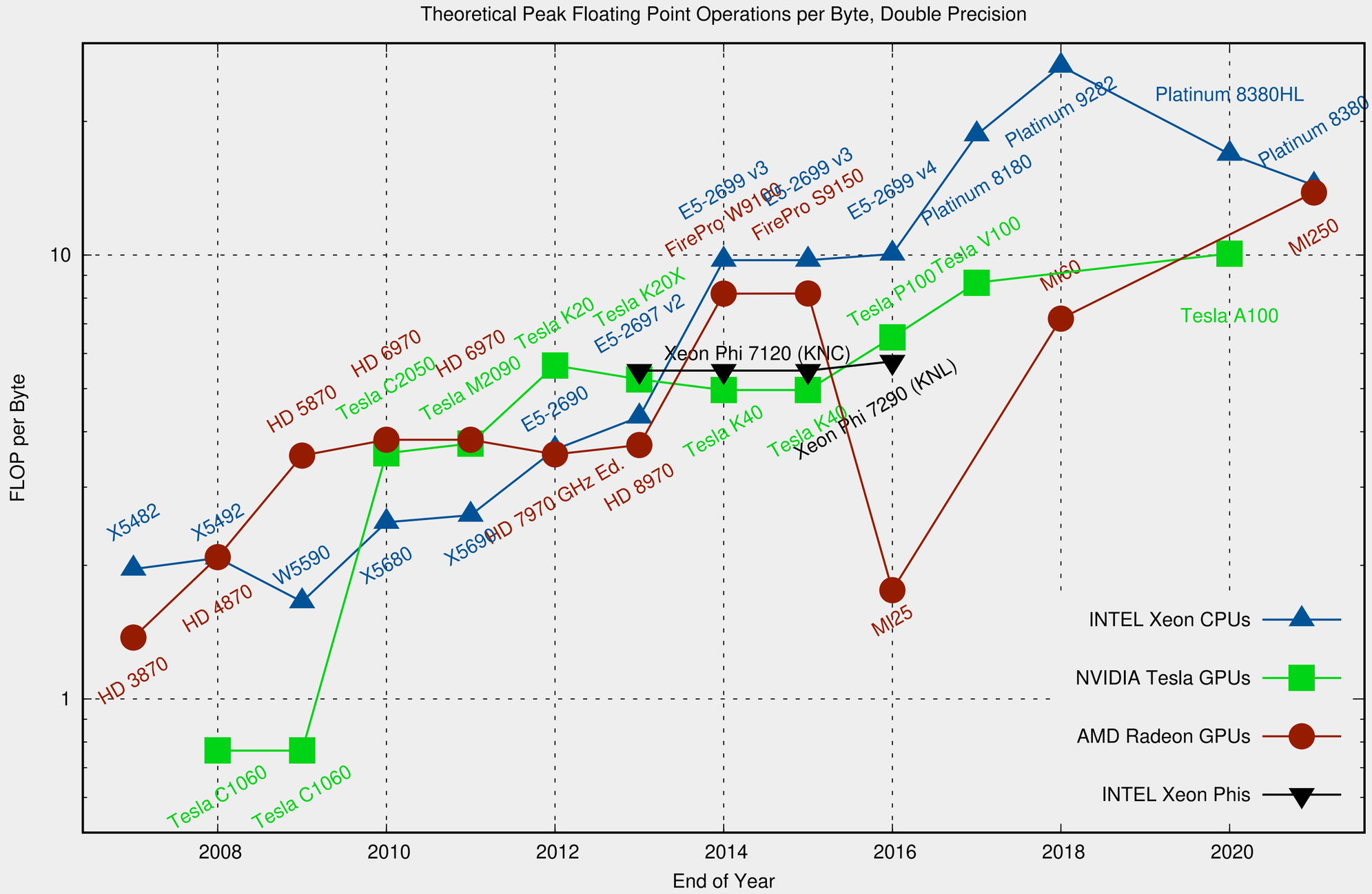
- High order doesn't help because real problems have singularities.
- Matrix-free is just for (very) high order



Approximation constants are good for high order

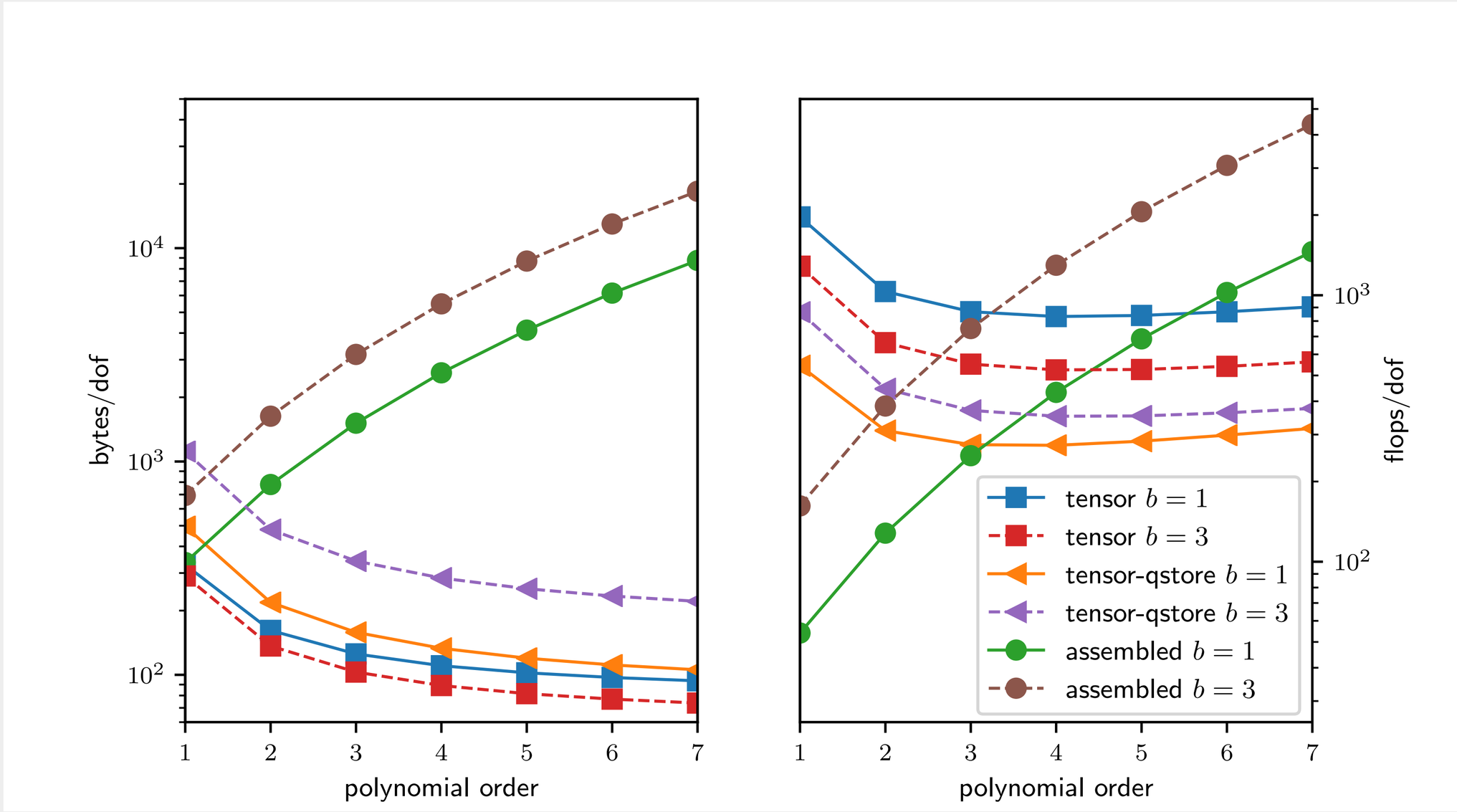


Bandwidth is scarce compared to flops

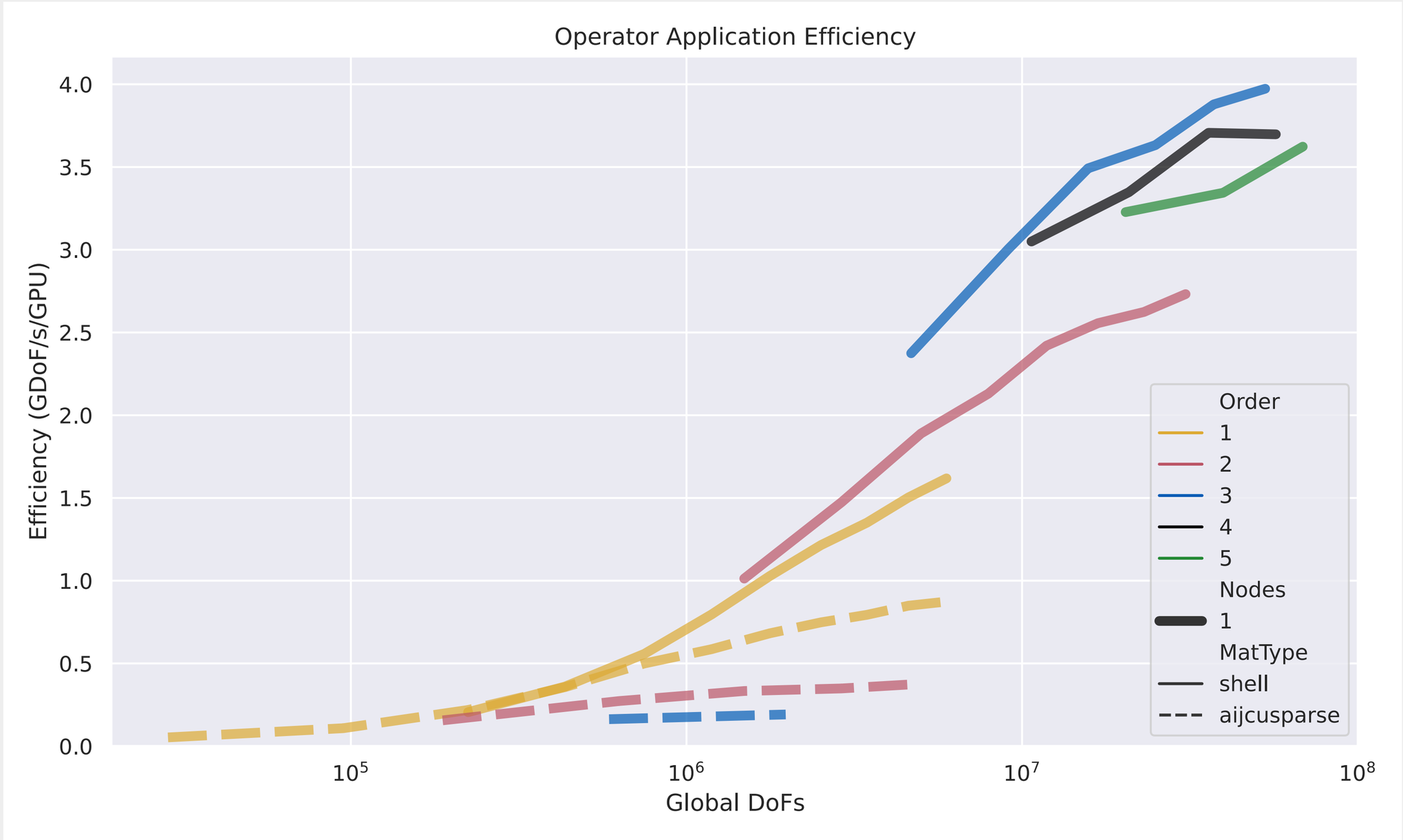


Why matrix-free?

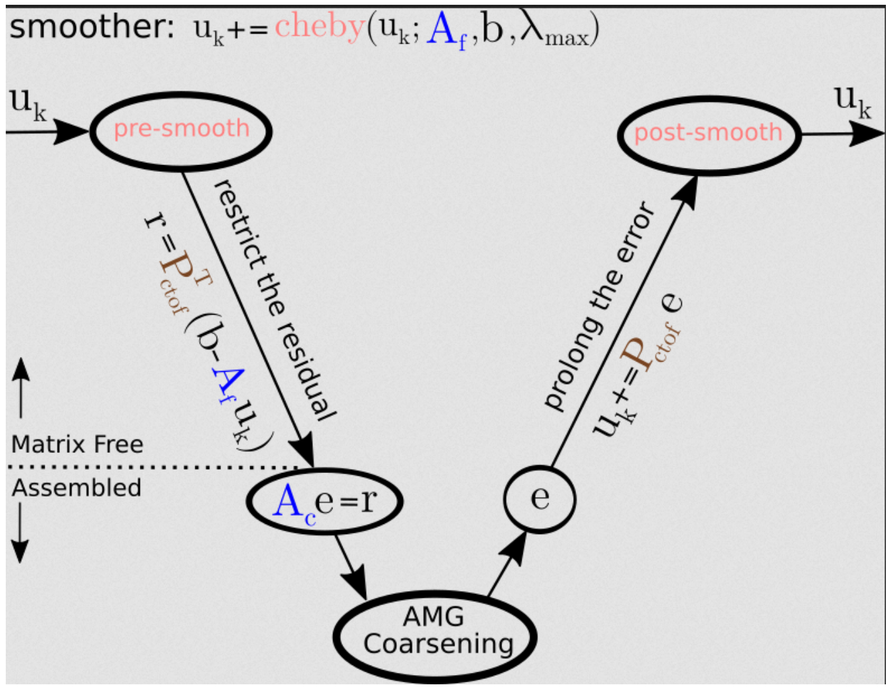
- Assembled matrices need at least 4 bytes transferred per flop. Hardware does 10 flops/byte. Matrix-free methods store and move less data, compute faster.

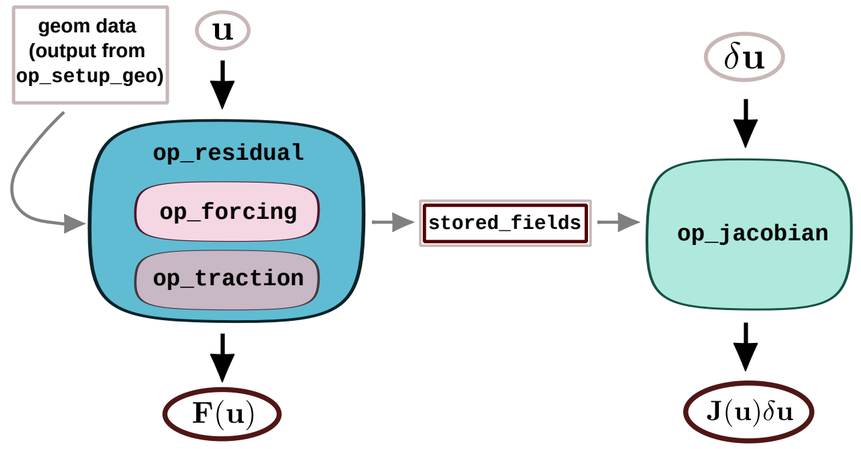


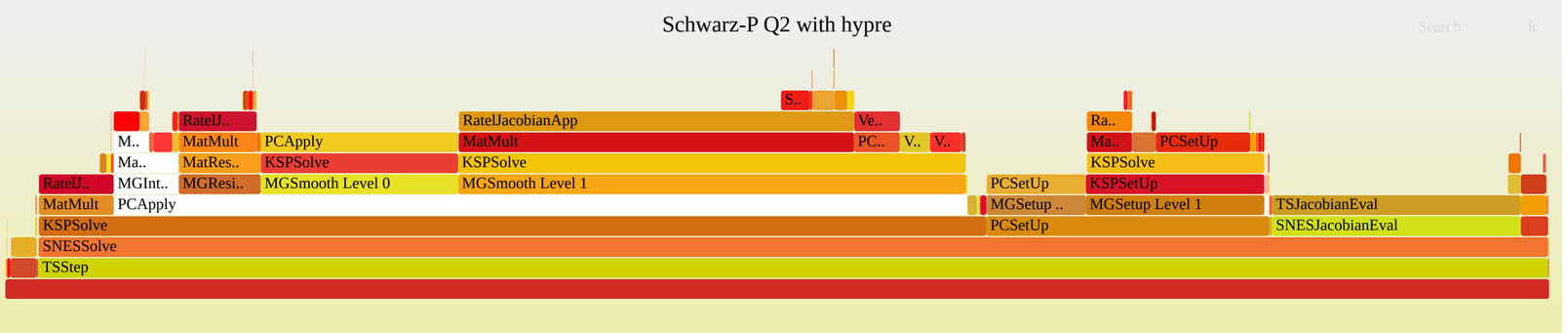
Matrix-free is already faster for Q_1 elements



p-multigrid algorithm and cost breakdown

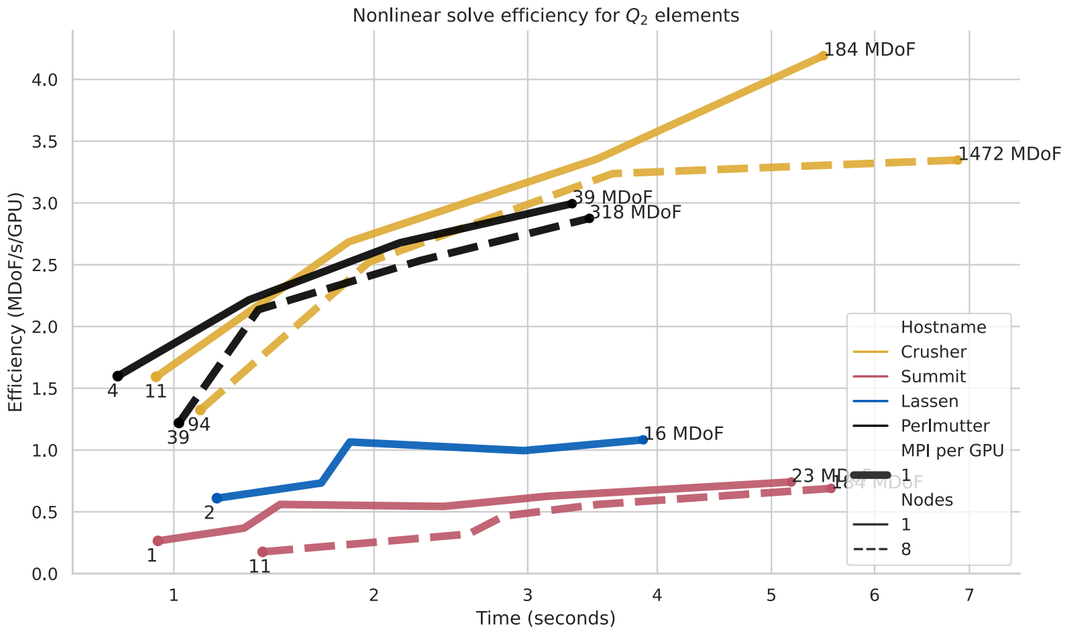




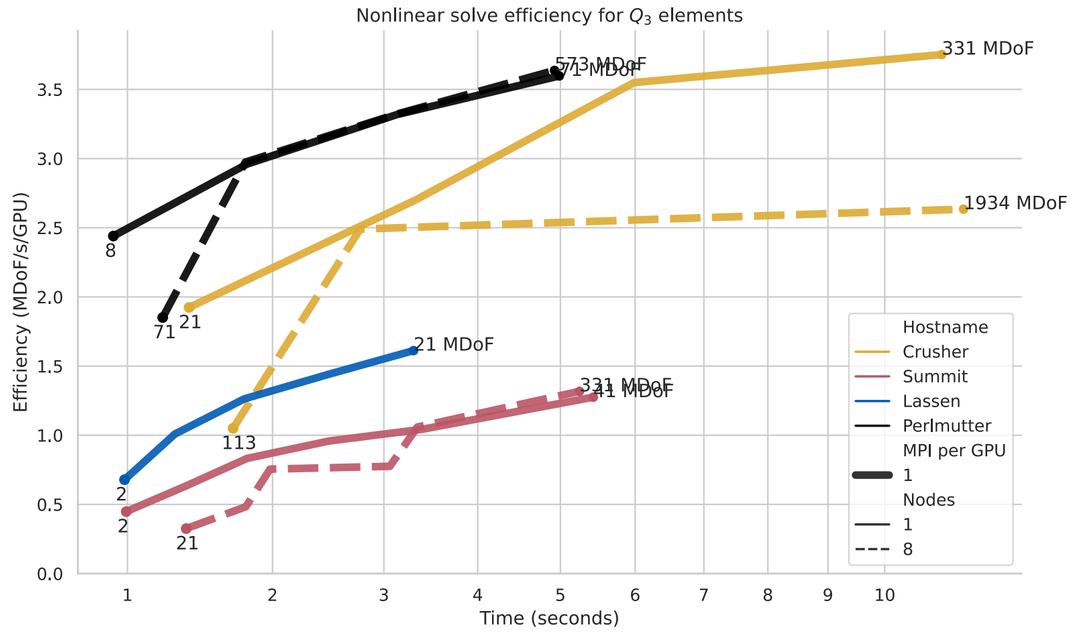


Nonlinear solve efficiency

Q_2 elements

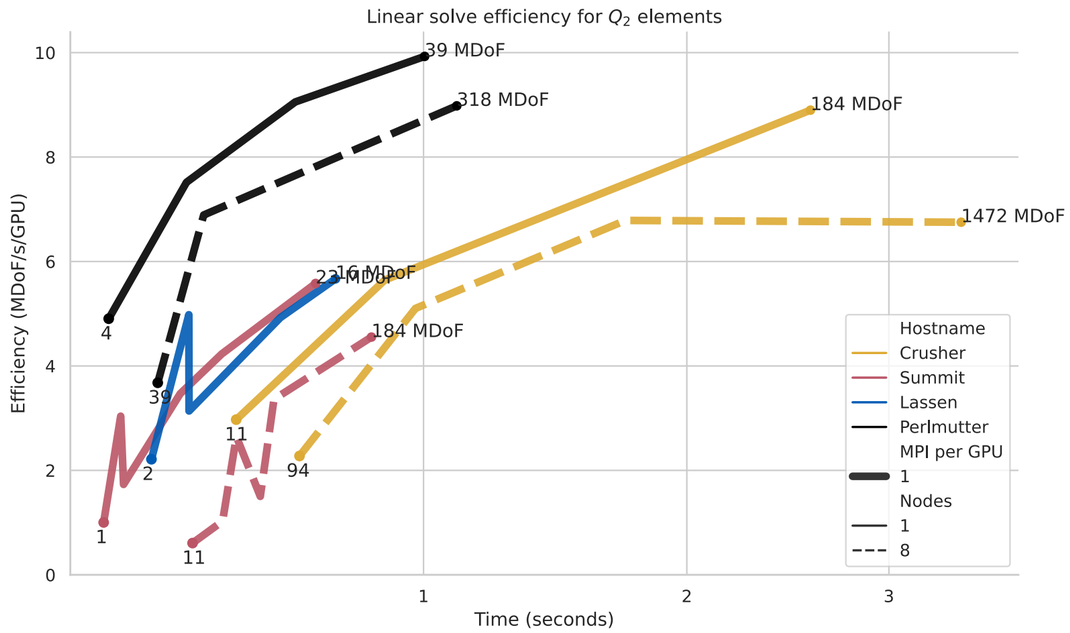


Q_3 elements

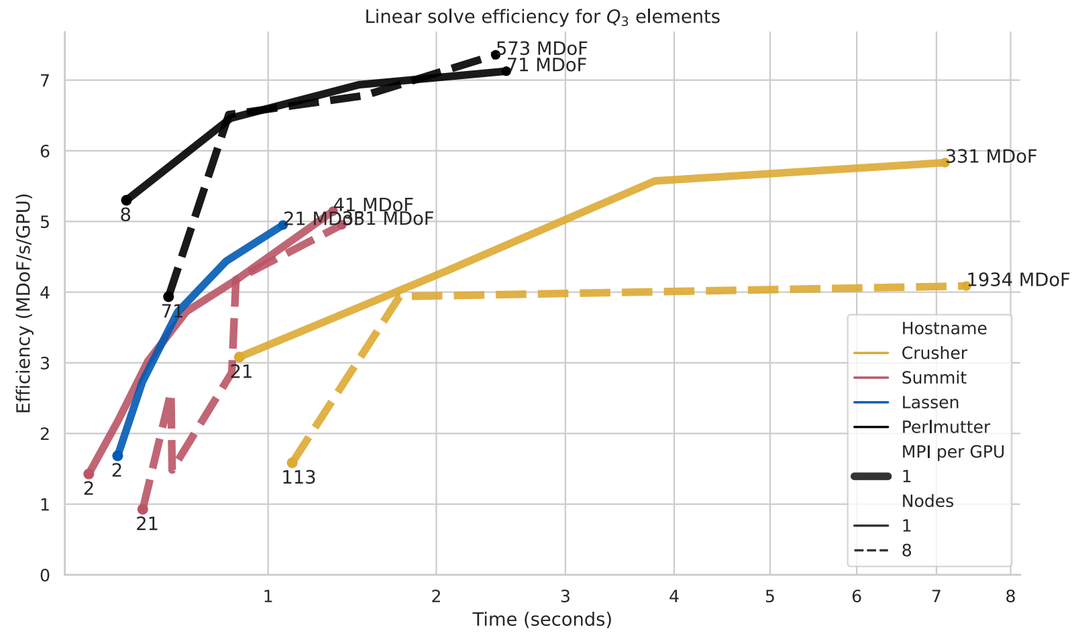


Linear solve efficiency

Q_2 elements



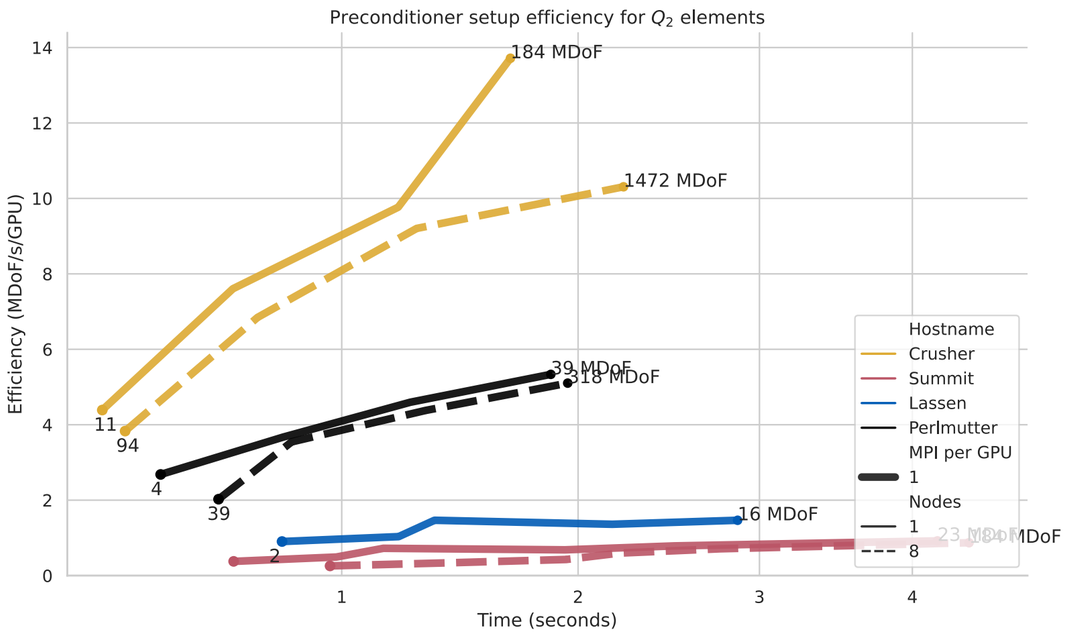
Q_3 elements



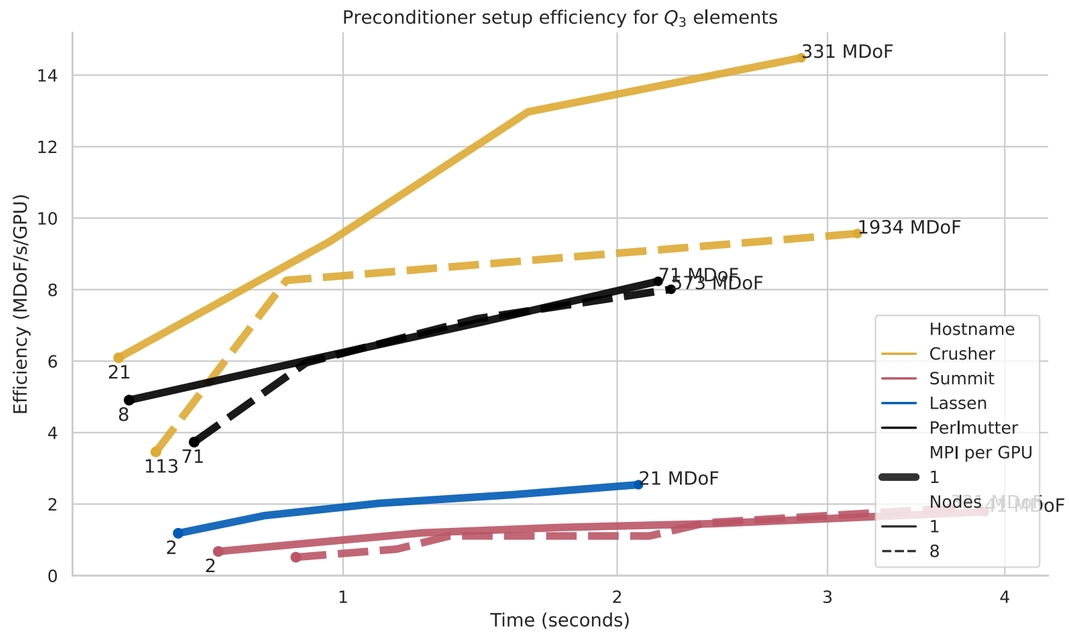
- Coarse solver is hypre BoomerAMG tuned configured for elasticity; thanks Victor Paludetto Magri.

Preconditioner setup efficiency

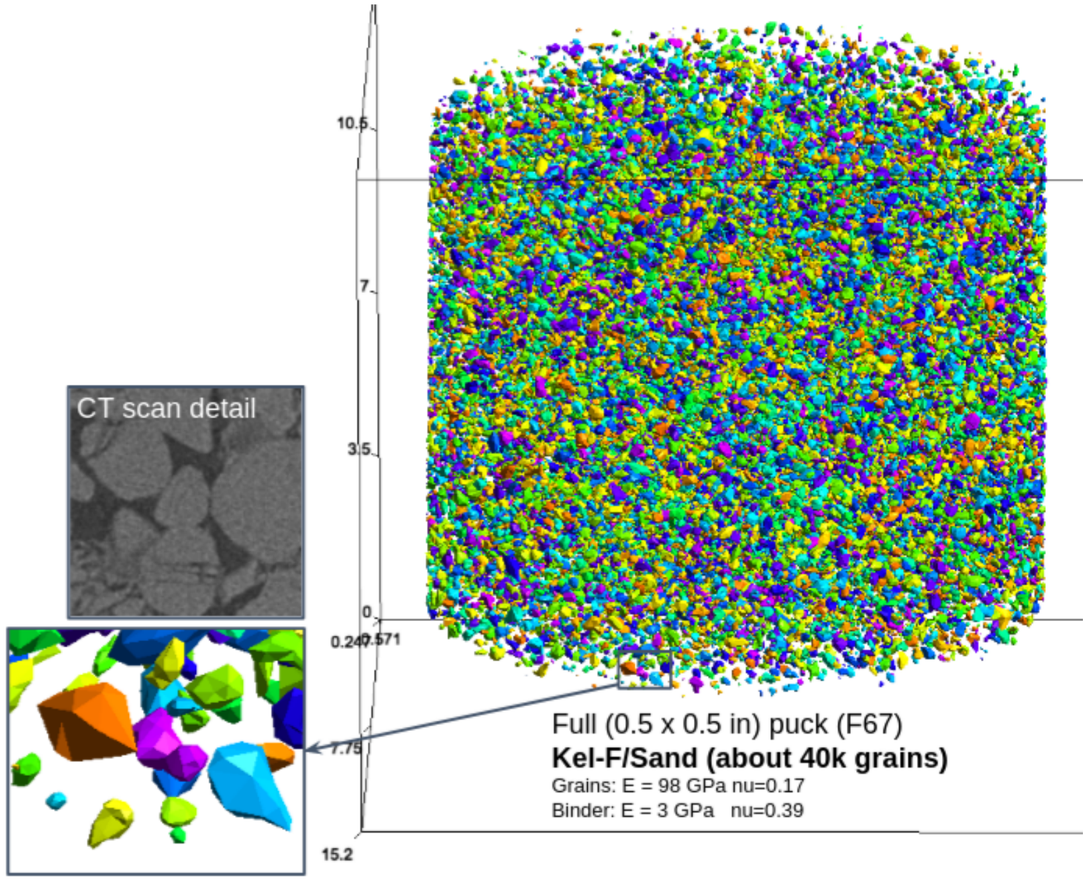
Q_2 elements



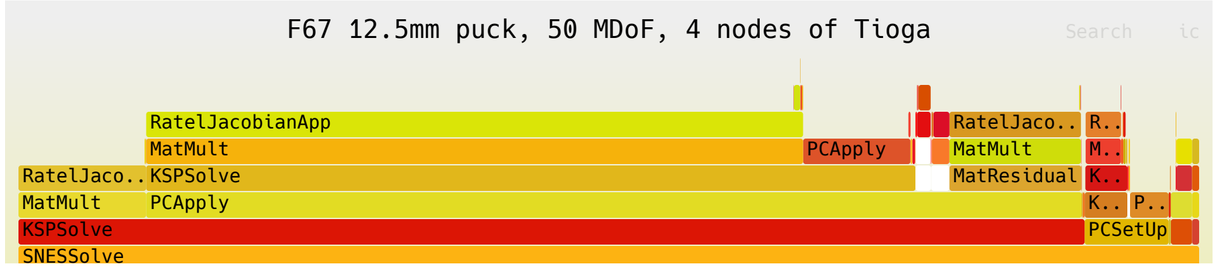
Q_3 elements



Half-inch puck (F67), 50 MDoF, quadratic tets



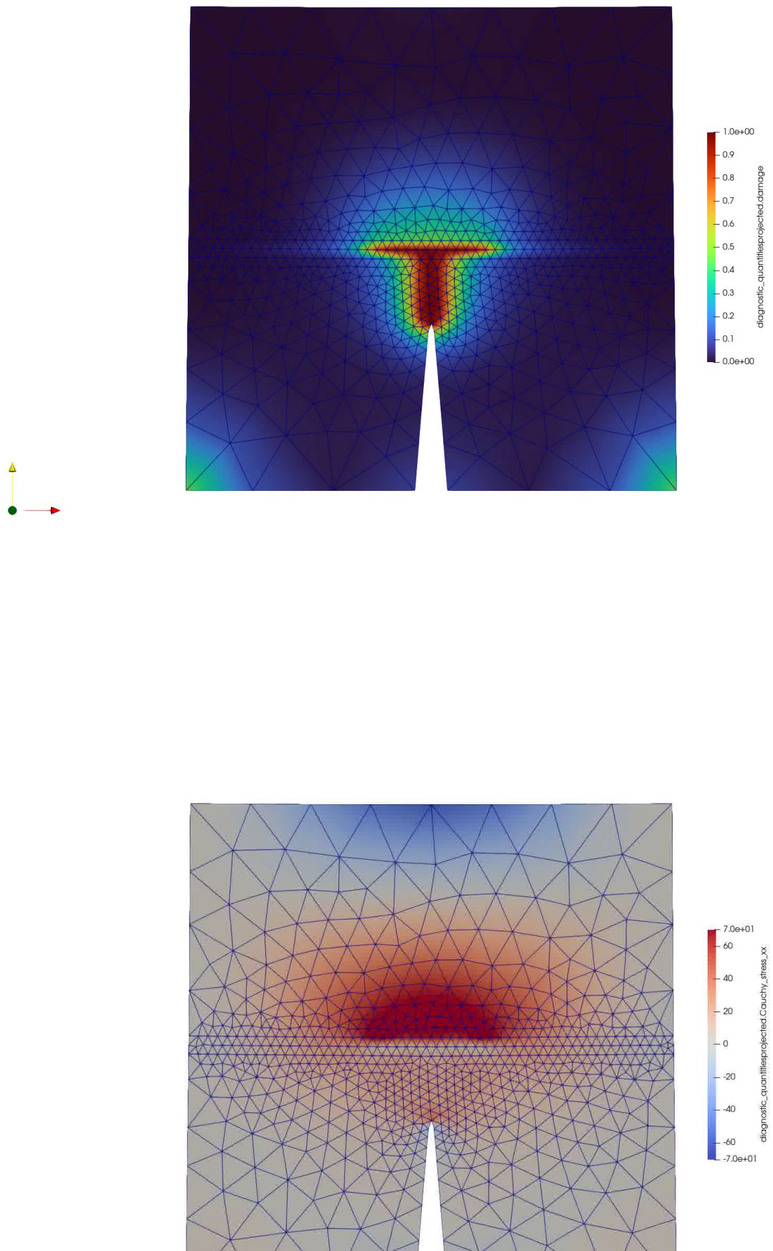
- 40k grains segmented from CT scans
- 1% global strain, neo-Hookean model
- 34 seconds per nonlinear solve (`rtol=1e-8`)
  - 7 seconds per linear solve
  - 45 CG iterations
- `/gpu/hip/shared` backend since `hip/gen` does not yet support tensor product elements
  - Will try `hip/magma`
- BoomerAMG coarse solve (linear elements)
- Pure-GPU assembly into hypre ParCSR



Phase-field damage mechanics

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \phi \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

- A is elasticity operator
- D is screened Laplacian for damage (Green's functions decay in a few elements)



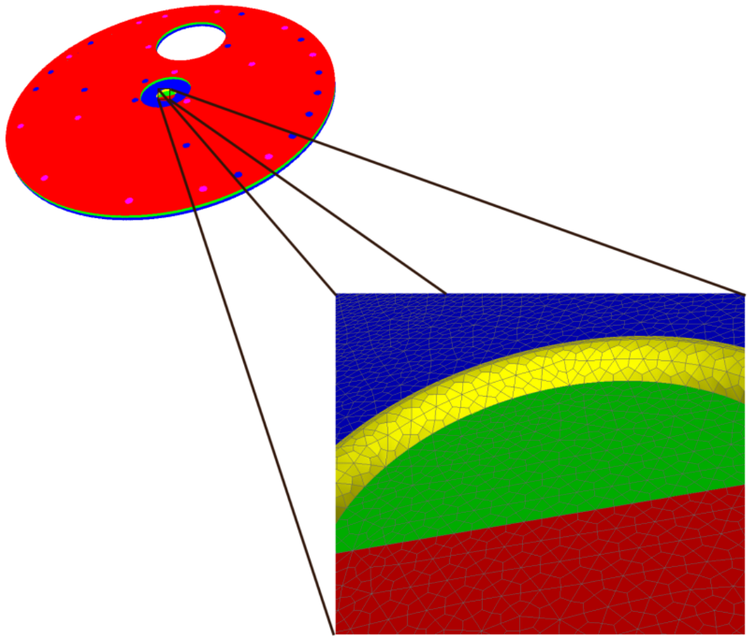
p-MG setup

- p-MG coarsen from quadratic to linear elements (tets in this example)
- specify 6-dimensional rigid body modes as near null space
- damage field ϕ is not needed in AMG
- optional: point-block Jacobi smoothing

One node of Crusher vs historical Gordon Bell

- 184 MDoF Q_2 elements nonlinear analysis in seconds

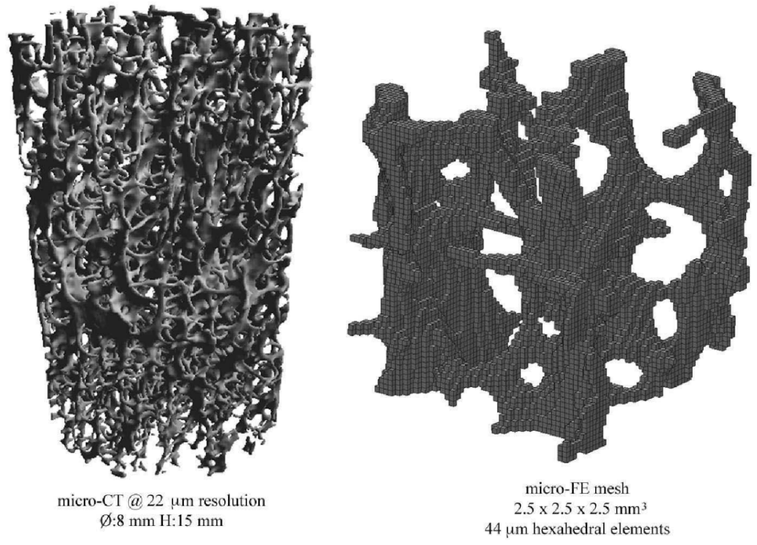
2002 Gordon Bell (Bhardwaj et al)



FE model	Size	N_p	Solution Time	Performance Rate	Overall CPU Efficiency
M2	110,000,000 dofs	3,783	418 seconds	745 Gflop/s	13%

Table 1
Static analysis of the optical shutter on 3,783 ASCI White processors: performance of Salinas for the FE model M2.

2004 Gordon Bell (Adams et al)



micro-CT @ 22 μm resolution
Ø:8 mm H:15 mm

micro-FE mesh
2.5 x 2.5 x 2.5 mm³
44 μm hexahedral elements

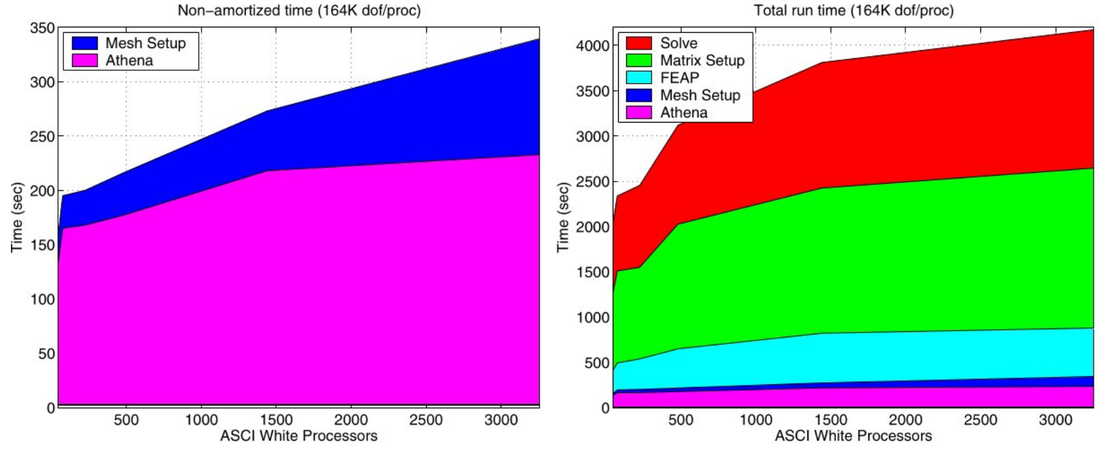
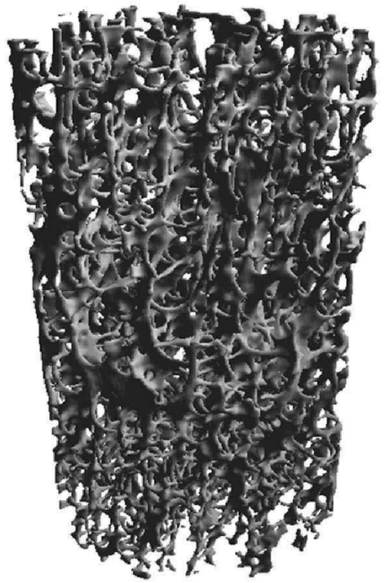
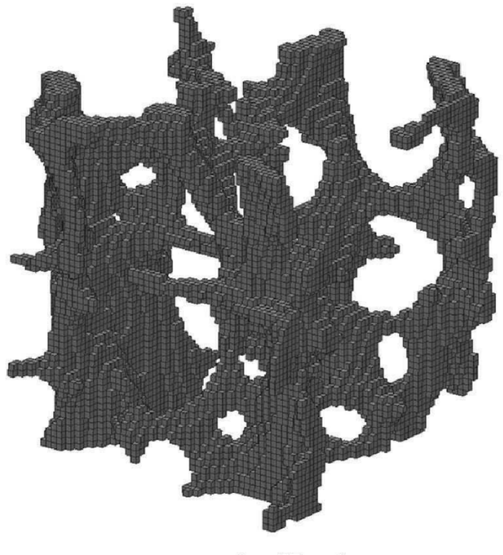


FIGURE 5.6. *Total end-to-end solution times - 164K dof per processor*

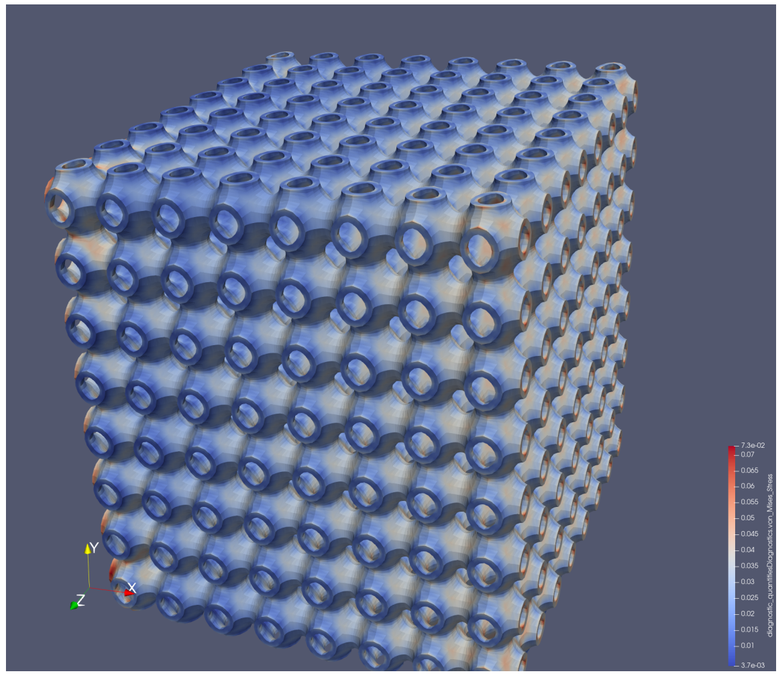
Comparison



micro-CT @ 22 μm resolution
Ø:8 mm H:15 mm

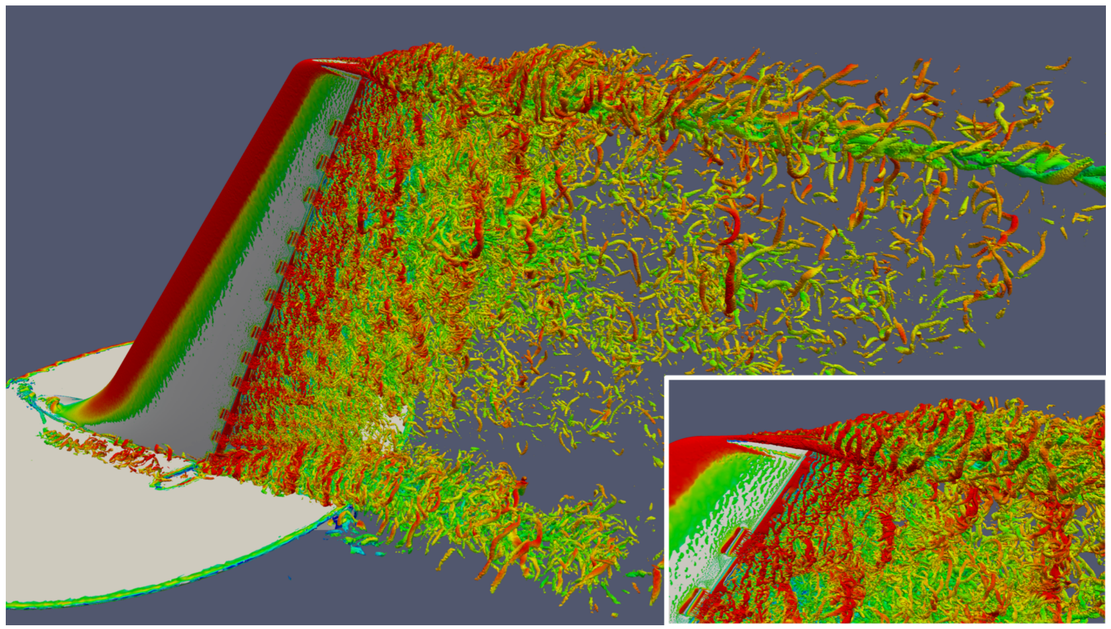


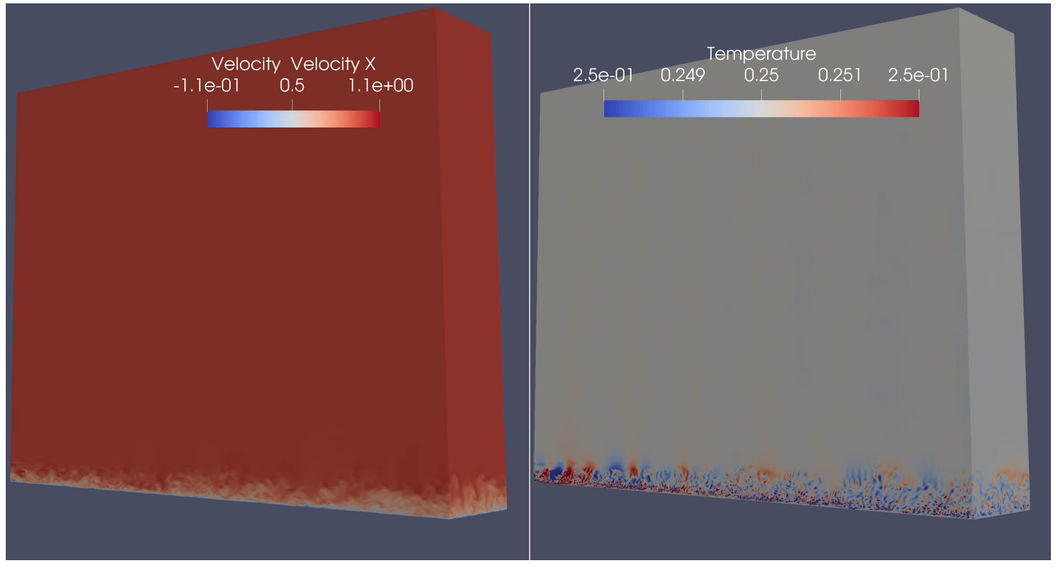
micro-FE mesh
2.5 x 2.5 x 2.5 mm³
44 μm hexahedral elements



Metric	Adams et al 2004	Ratel	Ratel
Discretization	linear	quadratic	cubic
Machine	ASCI White 130 nodes	Crusher 1 node	Crusher 1 node
Peak Bandwidth	1.56 TB/s	12 TB/s	12 TB/s
Degrees of freedom	237 M	184 M	331 M
kDoF/GB	460	400	700
load step strain	0.5%	12%	12%
kDoF/s per load step	600	6000	5500

Same story for compressible turbulence





PHASTA (Fortran)

- Extreme-scale unstructured CFD
- SUPG, implicit (gen-α) Newton-Krylov
- Aurora ESP: 2y on the "Intel/ALCF plan"
  - GPU still slower than CPU

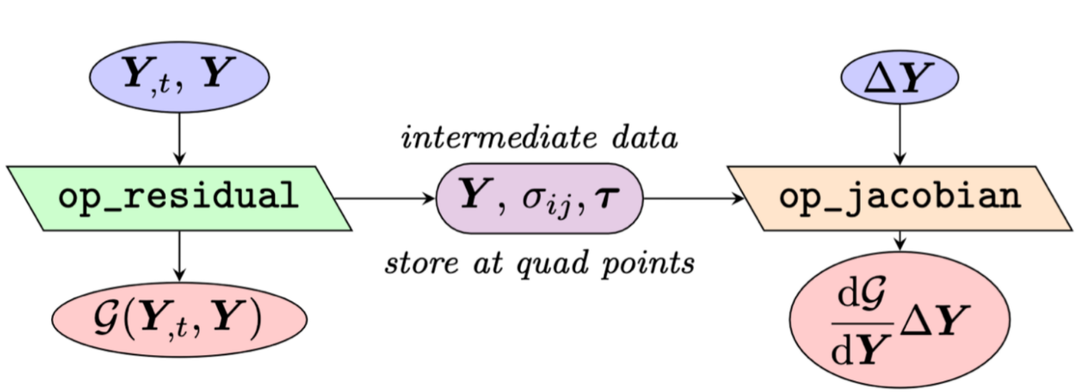
CEED-PHASTA

- All-new code, using libCEED with PETSc
- Matrix-free cuts setup/helps strong scaling
- End-to-end GPU (NVIDIA, AMD, Intel)

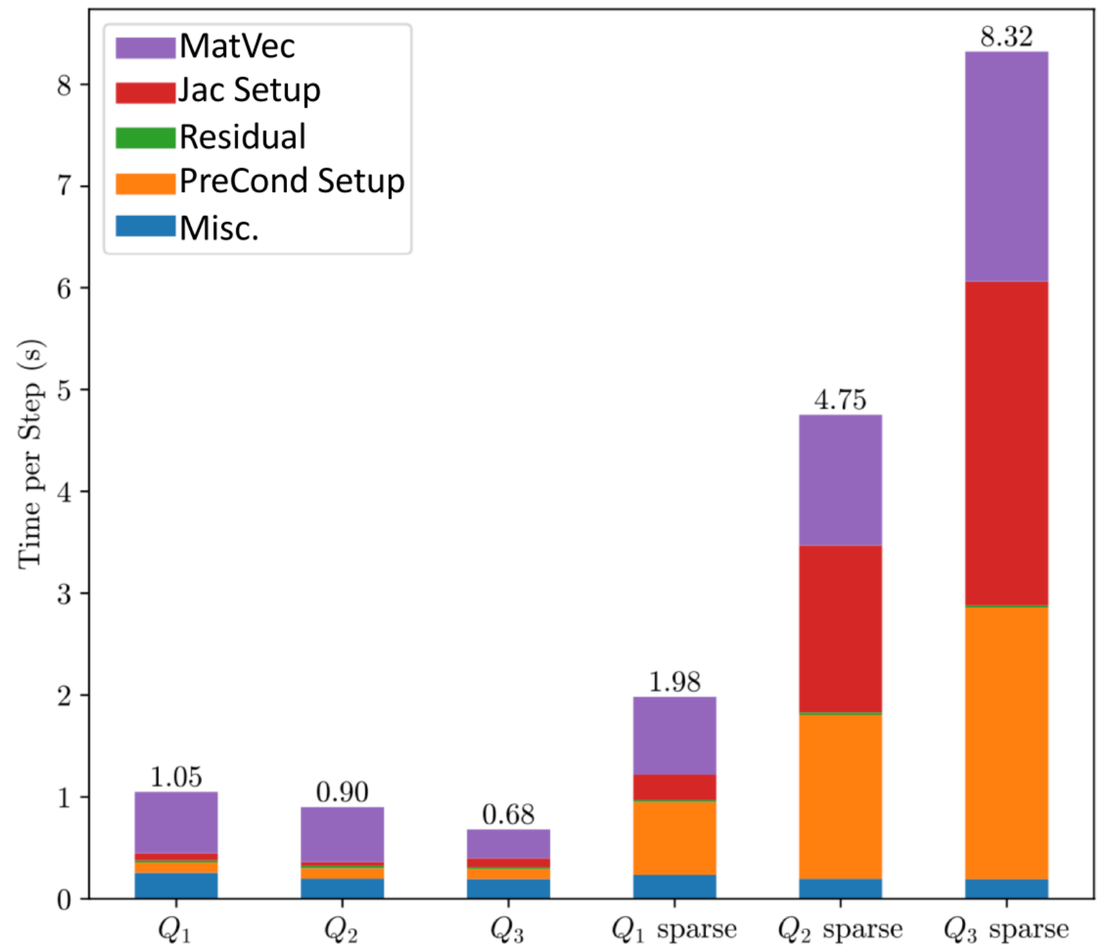
Code	Arch	Element	second/step
PHASTA	Skylake	Q_1	6-12
CEED	A100	Q_1	1.0
CEED	A100	Q_2	0.7
CEED	A100	Q_3	0.5

Algorithmic framework

- SUPG/VMS for compressible NS in pressure-primitive variables
- Implicit integration using gen-α
- 3 Newton iterations per time step
    - First two are very cheap (5-15 Krylov iterations), third is stiffer

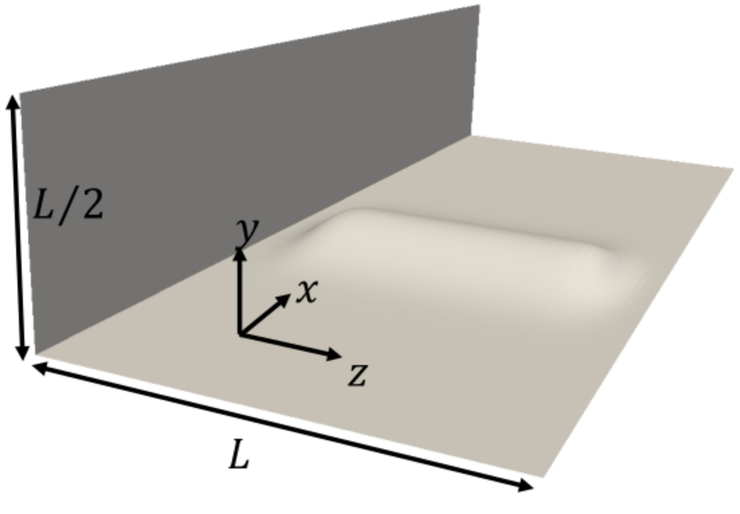


- Benchmark
    - flat plate $Re_\theta \approx 970$ STG inflow
    - $Ma \approx 0.1$
    - 12-30 nominal span/steamline resolution (plus units)



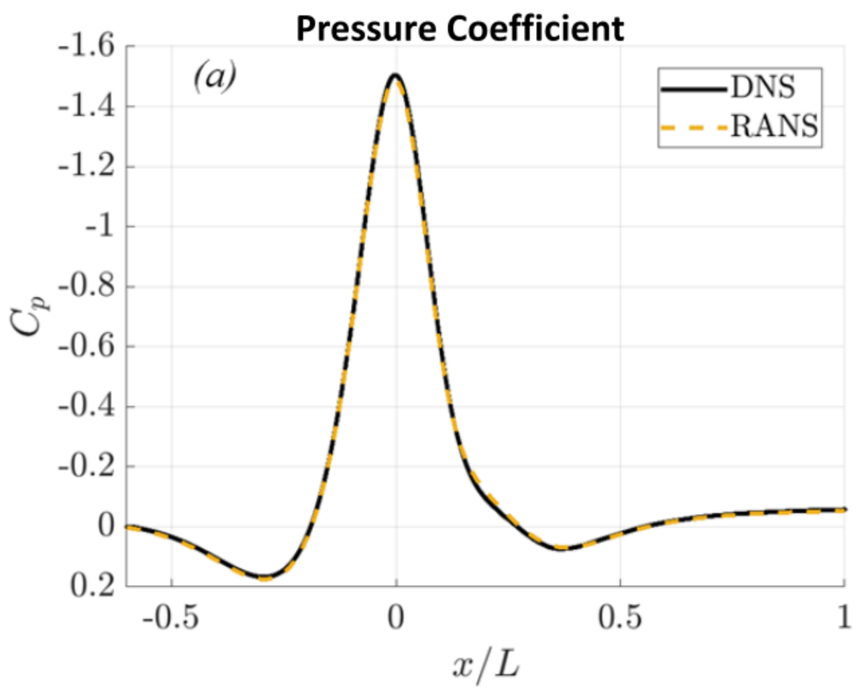
- 10 nodes of Polaris (4x A100/node)
- 250k nodes (1.25 MDoF) per GPU

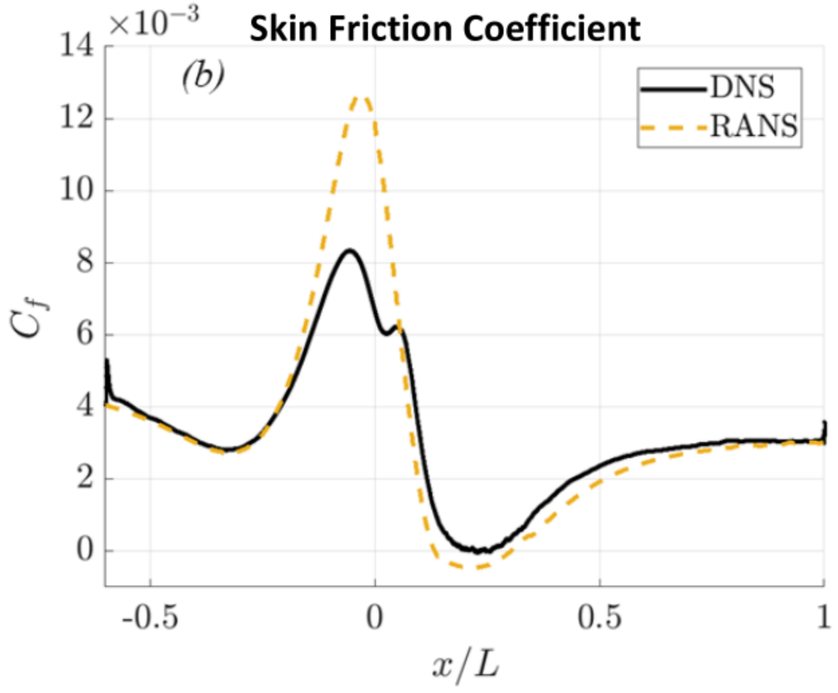
Boeing Speed Bump



- "Easy" problem for which RANS prediction of separation is catastrophic.
- Good experimental data available

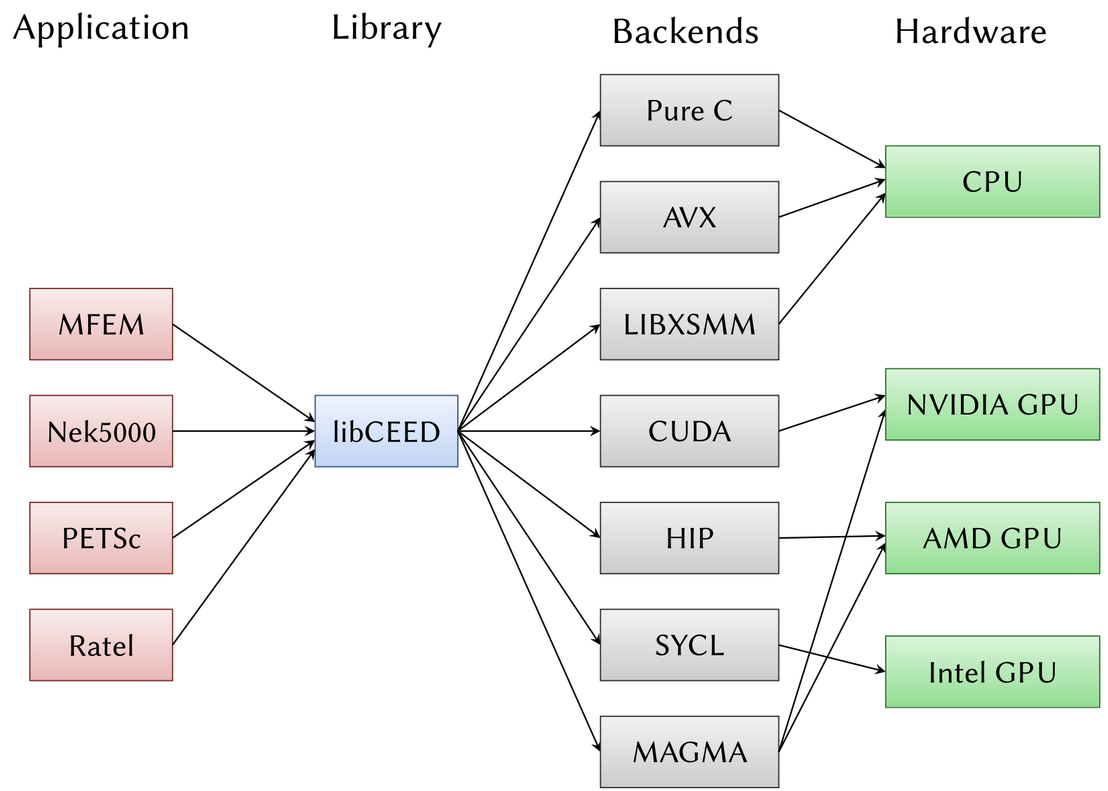
Can a RANS model predict a high-lift flow for the right reasons?





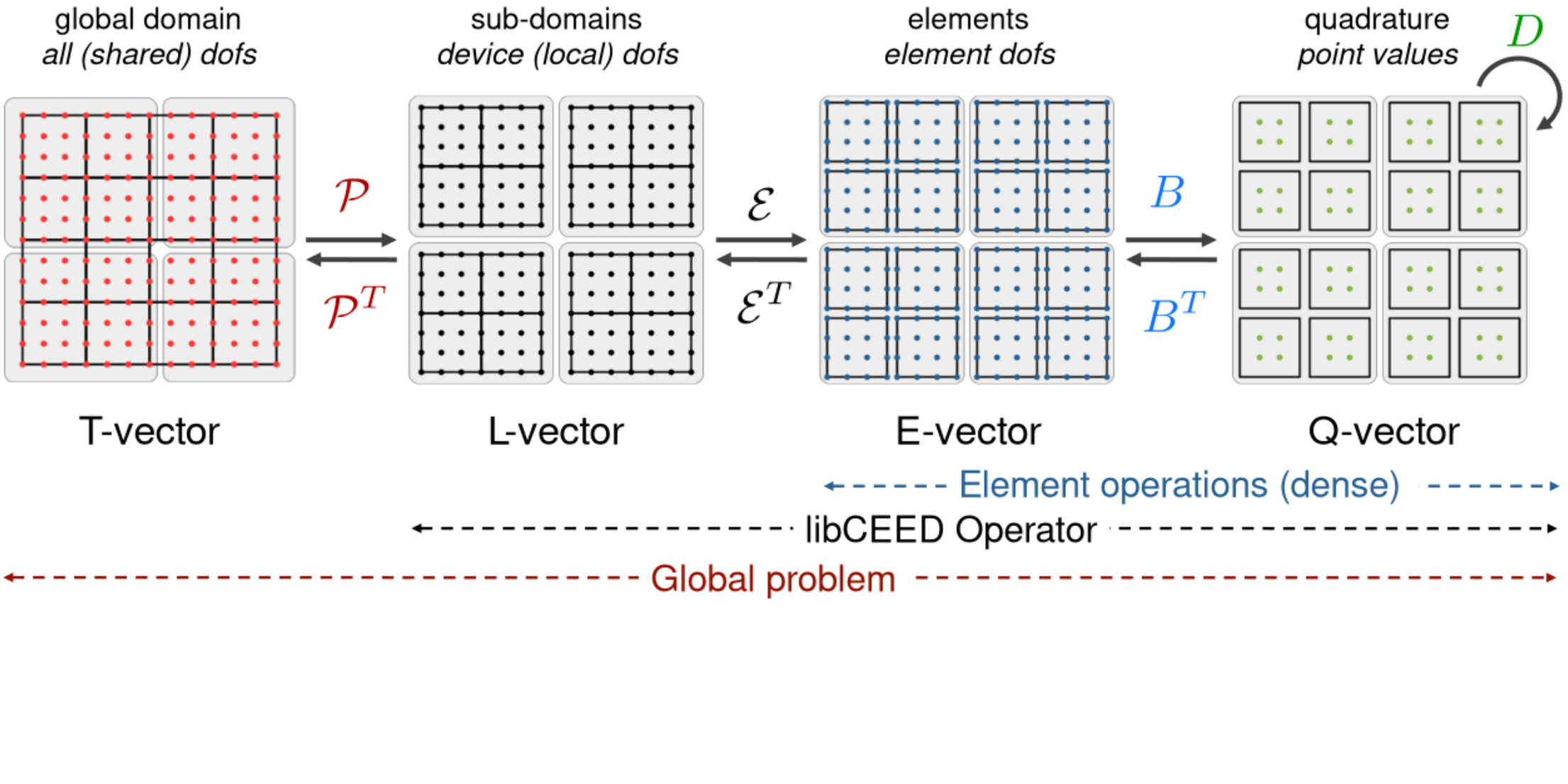
libCEED: fast algebra for finite elements

- Backend plugins with run-time selection
  - debug/memcheck, optimized
  - libxsmm, CUDA, HIP, SYCL
  - MAGMA to CUDA, HIP, SYCL*
- Single source vanilla C for QFunctions
  - Easy to debug, understand locally
  - C++ available, but not necessary
  - Target for DSLs, AD
- Python, Julia, Rust
- 2-clause BSD
- Available via MFEM, PETSc, Nek5000



Thanks to many developers, including Jeremy Thompson, Yohann Dudouit, Valeria Barra, Natalie Beams, Ahmad Abdelfattah, Leila Ghaffari, Will Pazner, Thilina Ratnayaka, Tzanio Kolev, Veselin Dobrev, David Medina

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



Quadrature functions: the math

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

$$v^T J w \sim \int_\Omega \begin{bmatrix} v \\ \nabla v \end{bmatrix}^T \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} w \\ \nabla w \end{bmatrix}$$

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

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

- B_I and B_∇ are tensor contractions -- independent of element geometry
- Choice of how to order and represent gathers $\mathcal{E}$ and scatters $\mathcal{E}^T$
- Similar for Neumann/Robin and nonlinear boundary conditions

QFunctions: debuggable, vectorizable, and JITable

- Independent operations at each of `Q` quadrature points, order unspecified

```
int L2residual(void *ctx, CeedInt Q,
    const CeedScalar *const in[],
    CeedScalar *const out[]) {
  const CeedScalar *u = in[0], *rho = ir
[1], *target = in[2];
  CeedScalar *f = out[0];
  for (CeedInt i=0; i<Q; i++)
    // Weak form of the problem goes her
    f[i] = rho[i] * (u[i] - target[i]);
  return 0;
```

$$\int v \, \underbrace{\rho (u - \mathtt{target})}_{f} = 0, \quad \forall v$$

```
      |   dp[j][k] = grad_du_tau[j][k] + 2*(mu - lam_log_J[0][i])*depsilon[j][k];
 1.65 |     mov     rdi,QWORD PTR [rsp+0x160]
      |   grad_du_tau[j][k] += grad_du[j][m]*temptau[m][k];
      |     vmovapd YMMWORD PTR [rsp+0x2e8],ymm6
      |   {tau[4][i], tau[3][i], tau[2][i]}
 0.08 |     vmovapd YMMWORD PTR [rsp+0x268],ymm1
      |   grad_du_tau[j][k] += grad_du[j][m]*temptau[m][k];
 0.16 |     vmovapd ymm1,ymm12
      |     vmulpd  ymm12,ymm5,ymm6
 0.08 |     vmulpd  ymm6,ymm10,YMMWORD PTR [rsp+0x308]
 1.66 |     vmovapd YMMWORD PTR [rsp+0x2a8],ymm1
      |     vfmadd231pd ymm6,ymm3,YMMWORD PTR [rsp+0x2c8]
 1.14 |     vfmadd231pd ymm12,ymm11,ymm1
 0.16 |     vmovapd ymm1,YMMWORD PTR [rsp+0x2e8]
      |     vfmadd231pd ymm14,ymm10,ymm11
```

Example QFunctions

- Riemann problems
- Return mapping for plasticity
- Nitsche contact

Why not a domain-specific language (DSL)?

Developer experience

- Indexed, refactoring tools
- Libraries of materials
- Unit testing, property testing
- Debugger integration
    - Run in debugger with `-fp_trap`, see how your code computed a negative pressure.
    - Attach debugger to running job, see why return-mapping algorithm is converging slowly.
- Static analysis
- Performance transparency
    - Profiling tools, flamegraph reflects source

Rust

- Excellent error messages.
- Guaranteed type- and memory-safety.
- Excellent tooling and libraries
- `no_std`: compiling for the host ensures no allocation/system access (that would fail on device)
- Zero-cost FFI: JIT fuse kernels with CUDA-C parts and Rust parts; result is fully inlined.
- Ergonomic and safe AD via Enzyme
    - Working to merge upstream for `+nightly`

Modeling principles for matrix-free methods

Seek well-conditioned formulations

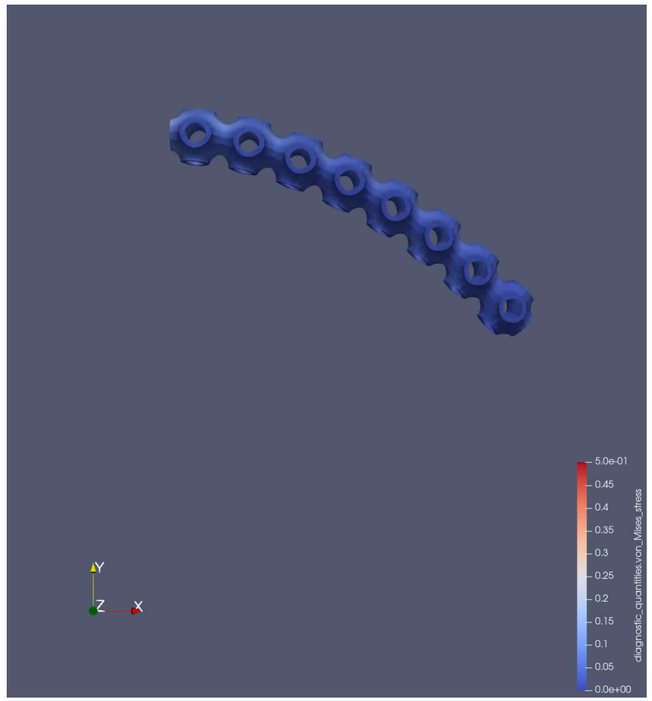
- Nitsche contact vs Lagrange multipliers or penalties
- Conforming discretizations vs XFEM and immersed boundary
- Mixed FE vs displacement-only elasticity

Smooth everything

- Leave extra degrees of freedom in
  - Skip static condensation
- Approx Braess-Sarazin vs segregated MG vs Vanka vs vertex-star
- "optimal" asymptotics must be weighted against implementation efficiency

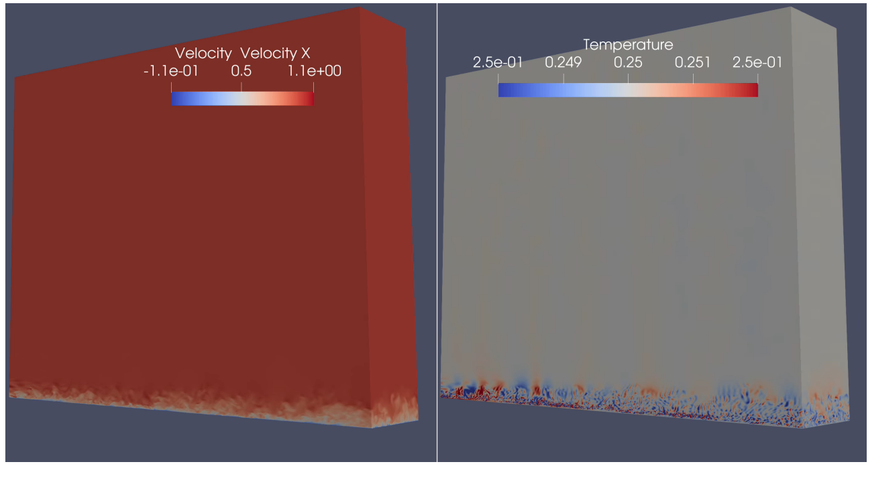
Outlook: petsc.org libceed.org ratel.micromorph.org

- You can move from Q_1 to Q_2 elements for about 2x cost (despite 8x more DoFs); $p > 2$ is free
- Mesh to resolve geometry, p-refine to pragmatic accuracy (tools!)
- libCEED already offers 2x speedup for Q_1
- Gordon Bell scale from 20 years ago ↦ interactive on a workstation (if you can buy MI250X 😊)



Come to the hands-on session

- Run p-MG solvers for structural mechanics on Tioga
- Explore QFunctions in real code
- Discuss unstructured implicit discretization and solvers



Old performance model

Iterative solvers: Bandwidth

- SpMV arithmetic intensity of 1/6 flop/byte
- Preconditioners also mostly bandwidth
    - Architectural latency a big problem on GPUs, especially for sparse triangular solves.
    - Sparse matrix-matrix products for AMG setup

Direct solvers: Bandwidth and Dense compute

- Leaf work in sparse direct solves
- Dense factorization of supernodes
    - Fundamentally nonscalable, granularity on GPUs is already too big to apply on subdomains
- Research on H-matrix approximations (e.g., in STRUMPACK)

New performance model

Still mostly bandwidth

- Reduce storage needed at quadrature points
    - Half the cost of a sparse matrix already for linear elements
    - Big efficiency gains for high order
- Assembled coarse levels are much smaller.

Compute

- Kernel fusion is necessary
- Balance vectorization with cache/occupancy
- $O(n)$, but benefits from BLIS-like abstractions

BLIS	libCEED
packing	batched element restriction
microkernel	basis action
?	user-provided qfunctions