High-performance matrix-free operator application and preconditioning

Jed Brown (ANL and CU) jedbrown@mcs.anl.gov
Dave May (ETH Zürich) dave.may@erdw.ethz.ch
Matt Knepley (UChicago) knepley@ci.uchicago.edu

Download this poster from http://59A2.org/files/201405-MatrixFree.pdf

Don't assemble

The purpose of a solver is to solve equations. Assembly is a necessary evil only for those legacy algorithms that depend on assembled sparse matrices. The comfortable abstraction from a bygone era is now a performance bottleneck with teeth on both ends. Ignoring the time and memory cost to assemble a matrix, it can take longer to compute a residual using an assembled operator than to solve up to discretization error with a fast matrix-free method.

Sparse matrices are bad for hardware

Once assembled, matrices suffer from low arithmetic intensity, performing less than 1/4 flop for each byte of memory loaded into cache.

Table 1: Bandwidth and compute balance for modern hardware.

Processor	STREAM (GB/s)	Peak (GF/s)	Balance (F/B)
E5-2697v2 12-core	45	230	5.2
BG/Q 16-core	26	205	7.9
Kepler K20Xm	160	1310	8.2
Xeon Phi SE10P	161	1060	6.6

A lower-memory representation using more flops would use hardware more efficiently.

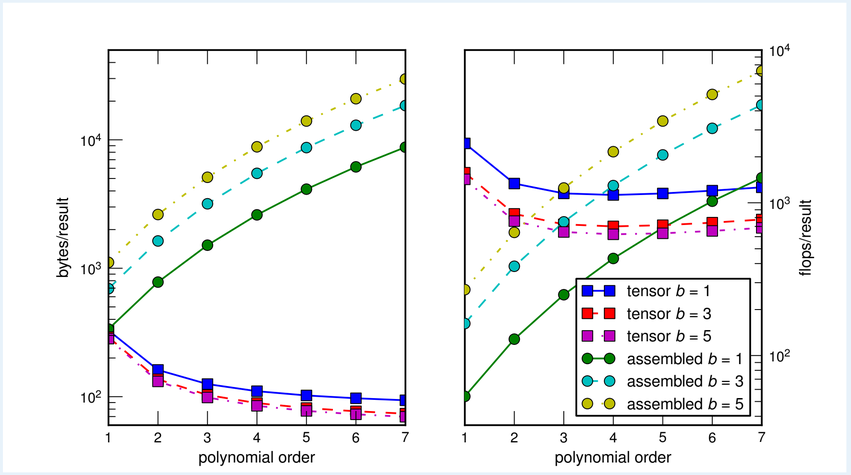

Figure 1: Cost per dof to apply an assembled matrix vs matrix-free tensor product for Q_k discretization in 3D. The operator defined as Newton linearization of nonlinear material model, stored at quadrature points. Unassembled uses hardware more efficiently and is asymptotically better for high polynomial order and for large block size b.

Coefficients often have structure imparted by a material model. For example, a scalar nonlinear diffusion operator $-\nabla \cdot \left[\kappa\left(\frac{1}{2}|\nabla u|^2\right), \nabla u\right]$ has Newton linearization $-\nabla \cdot \left[(\kappa \mathbf{1} + \kappa' \nabla u \otimes \nabla u)\nabla w\right]$ so it is sufficient to store the 4 values $\{\kappa, \sqrt{\pm\kappa'}\nabla u\}$ so long as κ' does not change sign [1].

$10\times$ speedup for Q_2 elements (Stokes or elasticity)

We know analytically (Figure 1) that unassembled wins for sufficiently high order, but what are the constants for a practical implementation? We consider operator application for a variable-viscosity Stokes problem on Q_2-isoparametrically mapped grids.

Table 2: Memory and compute demands per element for different operator application methods. Arithmetic intensity is reported as flops/byte, counting adds, multiplies, and division (rare) all as 1 flop. "Tensor" is a matrix-free implementation that uses the tensor product structure present on the reference element. "Tensor C" absorbs the metric terms into a tensor-valued coefficient. Average time (milliseconds) and GF/s is reported for (parallel) operator application on 8 nodes of Edison (3686 GF/s peak).

Operator	flops	Pessimal cache bytes	Pessimal cache F/B	Perfect cache bytes	Perfect cache F/B	Time (ms)	GF/s
Assembled	9216	—	—	37248	0.247	42	113
Matrix-free	53622	2376	22.5	1008	53	22	651
Tensor	15228	2376	6.4	1008	15	4.2	1072
Tensor C	14214	5832	2.4	4920	2.9	—	—

pessimal cache Each Q_2 element must be retrieved independently from DRAM.
perfect cache Each dof must be brought into cache only once.

Cache versus vectorization

Performance is limited by memory bandwidth (often with many memory streams), cache reuse, and vectorization. Vectorization across elements does not require cross-lane operations, but increases working set size relative to intra-element vectorization.
Intra-element vectorization performs well at high order (usually Q_3 and higher), but the optimizations are more sensitive to polynomial order and number of fields. Current trends of longer vector registers and more hardware threads per core (e.g., Xeon Phi, BG/Q, GPUs) effectively reduce the amount of cache available per vector lane. To realize high performance on such architectures, we either need rich cross-lane vector instructions (and compilers capable of using them) or cache sizes commensurate with the number of vector lanes made available.
Cache and prefetch streams can be shared between hardware threads by interleaving thread responsibility, but a work-partition in FEM is not a dof-partition, so this technique causes overlapping writes. Writes can be managed by coloring, partitioning with duplication of interface points (to be summed later), or atomics/locking. Coloring has poor memory locality and the "cache-friendly" interleaved ordering is a pessimal partition (causing memory bloat for duplication and frequent conflicted writes for atomics/locking).

Matrix-free operator application

Discretize $-\nabla\left(\kappa\nabla \cdot u\right)$, yielding

$$A\mathbf{u} = \sum_{e \in \text{Elements}} \mathcal{E}_e^T \mathcal{D}_{\mathbf{x}}^T \Lambda(\omega\kappa)\mathcal{D}_{\mathbf{x}}\mathcal{E}_e\mathbf{u} \quad (1)$$

The physical gradient matrix $\mathcal{D}_{\mathbf{x}} = \{\mathcal{D}_i | i \in \{x, y, z\}\}$ is an 81×27 matrix for Q_2 elements in 3D using 3^3-point Gauss quadrature. In most finite-element implementations, the precomputed reference gradient matrix $\mathcal{D}_\xi$ is mapped to physical space via the block-diagonal operation $\mathcal{D}_{\mathbf{x}} = \Lambda(\nabla_{\mathbf{x}}\xi)\mathcal{D}_\xi$. This is wasteful; it is less expensive to rearrange as

$$A\mathbf{u} = \sum_{e \in \text{Elements}} \mathcal{E}_e^T \mathcal{D}_\xi^T \Lambda\Big((\nabla_{\mathbf{x}}\xi)^T(\omega\kappa)(\nabla_{\mathbf{x}}\xi)\Big)\mathcal{D}_\xi\mathcal{E}_e\mathbf{u}. \quad (2)$$

where $\nabla_{\mathbf{x}}\xi = (\nabla_\xi \mathbf{x})^{-1}$ is computed at quadrature points from the coordinate gradients. In Table 2, "Matrix-free" implements Equation 1 in the traditional way, "Tensor" implements Equation 2 using

$$D_\xi = \{\hat{D} \otimes \hat{B} \otimes \hat{B}, \hat{B} \otimes \hat{D} \otimes \hat{B}, \hat{B} \otimes \hat{B} \otimes \hat{D}\} \quad (3)$$

where $\hat{D}$ and $\hat{B}$ are the 3×3 differentiation and basis evaluation matrices for one dimension, and "Tensor C" stores the tensor $(\nabla_{\mathbf{x}}\xi)^T(\omega\kappa)(\nabla_{\mathbf{x}}\xi)$ at quadrature points, trading storage for explicit handling of metric terms. Tensor representations are standard for spectral element methods with collocated quadrature, but are rare for low-order FE with accurate quadrature.

Assembly-free preconditioning

Fast operator application is nearly useless without effective preconditioning. Fortunately [2], multigrid methods with polynomial smoothing are effective for many elliptic problems. The end-to-end performance and memory benefit of matrix-free operator application is mostly realized on the finest level. For most applications, there is little penalty to using existing matrix-based methods on coarse grids, even when not carefully optimized. pTatin3D is a lithosphere dynamics package that simulates thermodynamics and other processes in visco-plastic quasi-incompressible materials using a material-point method to track post-failure composition. A P_1^{disc} pressure space is necessary for local conservation and to accurately represent the hydrostatic mode in the free-surface flows, thus a Q_2 velocity space is used to maintain uniform inf-sup stability. Solving the Newton step for the nonlinear Stokes problem is the dominant cost, and is achieved using either block-triangular preconditioners (Table 3) or Schur-complement reduction.

Table 3: pTatin3D efficiency for different problem and partition sizes on Edison.

Operator	Cores	Grid	El/core	Solve/core El/s	Solve/core GF/s	Op/core kEl/s
Assembled	192	64^3	1265	46	0.9	33
Matrix-free	192	64^3	1265	69	2.6	62
Tensor	192	64^3	1265	128	2.4	323
Matrix-free	1536	96^3	576	47	2.2	60
Tensor	1536	96^3	576	72	2.2	252
Tensor	12288	192^3	576	26	1.1	166

For smooth problems, full multigrid is the method of choice. HPGMG-FE (https://hpgmg.org) solves an elliptic problem using Q_2 elements on deformed meshes, reaching discretization error with one F-cycle using 3,1 Chebyshev smoothing. Convergence thus requires 5 fine-grid operator applications, for a total of about 6 work units. With the efficiency in Table 2, we see that the entire solve can be less expensive than a single residual evaluation using a pre-assembled matrix.

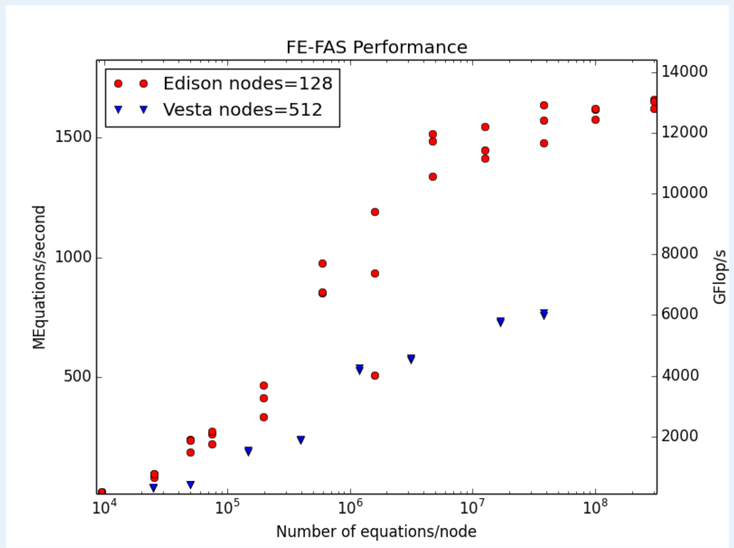

Figure 2: HPGMG-FE FMG performance as a function of per-node problem size on 128 nodes of Edison (Cray XC-30; up to 23% of peak) versus 512 nodes of Vesta (Blue Gene/Q; up to 6% of peak).

Outlook

Cache versus vectorization is fundamental: tradeoffs vary with element order and numbers of degrees of freedom. Research in robust matrix-free multigrid techniques are needed.

References

Jed Brown.
Efficient nonlinear solvers for nodal high-order finite elements in 3D.
Journal of Scientific Computing, 45:48–63, 2010.

M. F. Adams, M. Brezina, J. J. Hu, and R. S Tuminaro.
Parallel multigrid smoothing: polynomial versus Gauss–Seidel.
J. Comp. Phys., 188(2):593–610, 2003.

This work was supported by the Department of Energy Office of Advanced Scientific Computing Research.