# Efficient Implicitness

## Latency-Throughput and Cache-Vectorization Tradeoffs

**Jed Brown** jedbrown@mcs.anl.gov (ANL and CU Boulder)

Heterogeneous Multi-Core workshop, NCAR, 2014-09-17

This talk:
http://59A2.org/files/20140917-EfficientImplicitness.pdf

# Intro

- I work on PETSc, a popular linear and nonlinear solvers library
- Some users need fastest time to solution at strong-scaling limit
- Others fill memory with a problem for PETSc
- Sparse matrices are a dead end for memory bandwidth reasons
  - but heavily embraced by legacy code and enable algebraic multigrid
- We need to restructure algorithms, but how?
- What are the fundamental long-term bottlenecks?
- Worrisome trends
  1. Fine-grained parallelism without commensurate increase in caches
  2. Emphasizing vectorization over cache reuse
  3. High instruction latency to be covered by hardware threads

# Intro

- I work on PETSc, a popular linear and nonlinear solvers library
- Some users need fastest time to solution at strong-scaling limit
- Others fill memory with a problem for PETSc
- Sparse matrices are a dead end for memory bandwidth reasons
  - but heavily embraced by legacy code and enable algebraic multigrid
- We need to restructure algorithms, but how?
- What are the fundamental long-term bottlenecks?
- Worrisome trends
  1. Fine-grained parallelism without commensurate increase in caches
  2. Emphasizing vectorization over cache reuse
  3. High instruction latency to be covered by hardware threads

# Hardware Arithmetic Intensity

| Operation | Arithmetic Intensity (flops/B) |
|---|---|
| Sparse matrix-vector product | 1/6 |
| Dense matrix-vector product | 1/4 |
| Unassembled matrix-vector product | $\approx 8$ |
| High-order residual evaluation | $> 5$ |

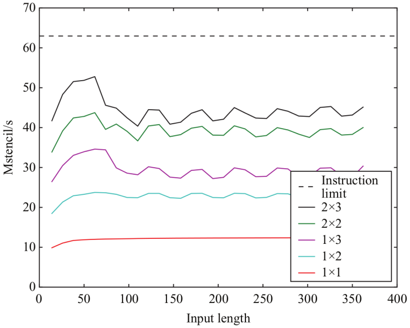
| Processor | Bandwidth (GB/s) | Peak (GF/s) | Balance (F/B) |
|---|---|---|---|
| E5-2680 8-core | 38 | 173 | 4.5 |
| E5-2695v2 12-core | 45 | 230 | 5.2 |
| Blue Gene/Q node | 29.3 | 205 | 7 |
| Kepler K20Xm | 160 | 1310 | 8.2 |
| Xeon Phi SE10P | 161 | 1060 | 6.6 |
| Haswell-EP (estimate) | 60 | 660 | 11 |
| KNL (estimate) | 100 (DRAM) | 3000 | 30 |

# How much parallelism out of how much cache?

| Processor | v width | threads | F/inst | latency | L1D | L1D/#par |
|---|---|---|---|---|---|---|
| Nehalem | 2 | 1 | 2 | 5 | 32 KiB | 1638 B |
| Sandy Bridge | 4 | 2 | 2 | 5 | 32 KiB | 819 B |
| Haswell | 4 | 2 | 4 | 5 | 32 KiB | 410 B |
| BG/P | 2 | 1 | 2 | 6 | 32 KiB | 1365 B |
| BG/Q | 4 | 4 | 2 | 6 | 32 KiB | 682 B |
| KNC | 8 | 4 | 4 | 5 | 32 KiB | 205 B |
| Tesla K20 | 32 | * | 2 | 10 | 64 KiB | 102 B |

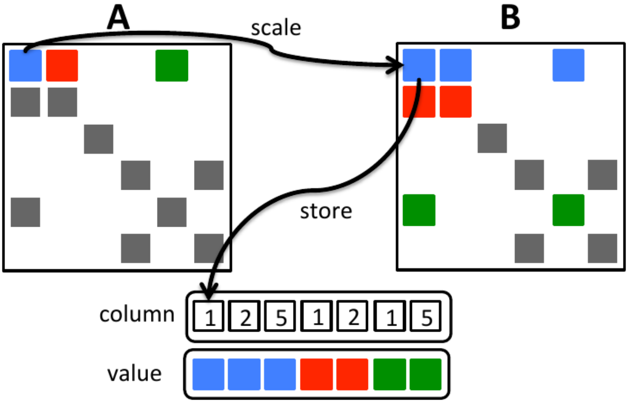
- Most “fast” algorithms do about $O(n)$ flops on $n$ data
- DGEMM and friends do $O(n^{3/2})$ flops on $n$ data
- Exploitable parallelism limited by cache and register load/store

# Story time: 27pt stencils instruction-limited for BG/P




- rolling 2-step kernel extended to 27-point stencil
- $2 \times 3$ unroll-and-jam used exactly 32 registers
- jam width limited by number of registers, barely covers ILP
- 200-entry jammed stream fits in L1
  - reuse in two directions for most problem sizes
- Malas, Ahmadia, Brown, Gunnels, Keyes (IJHPCA 2012)

# Fine-grained parallelism in SpMM




- Enumerate all scalar products contributing to row of product, $\hat{C}$
- Implemented using `scan` and `gather`
- Radix sort contributions to each row (two calls to `sort`)
- Contract row: `reduce_by_key`
- c/o Steve Dalton (2013 Givens Fellow, now at NVidia)

# CUSP Performance summary

| Matrix | CUSPARSE | Total Time Ref | Total Time Opt | |
|---|---|---|---|---|
| Cantilever | 61.9 | 57.6 | 21.6 | **2.8** / **2.7** |
| Spheres | 131.3 | 90.3 | 19.3 | **6.8** / **4.7** |
| Accelerator | 108.9 | 39.7 | 15.4 | **7.1** / **3.6** |
| Economics | 67.8 | 50.6 | 26.0 | **2.6** / **2.0** |
| Epidemiology | 72.3 | 57.0 | 17.4 | **4.2** / **3.3** |
| Protein | 92.0 | 56.2 | 39.4 | **2.3** / **1.4** |
| Wind Tunnel | 182.5 | 107.1 | 28.1 | **6.5** / **3.8** |
| QCD | 97.4 | 83.6 | 17.1 | **5.7** / **4.9** |
| Webbase | 3086.3 | 154.2 | 190.8 | **16.2** / **0.8** |

- New CUSP SpMM is faster than CUSPARSE for all test matrices.
- Sorting optimization faster except for very irregular graph.

# Memory overhead from expansion

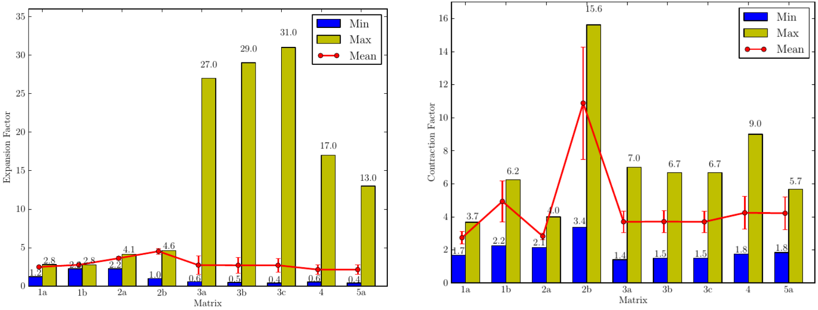
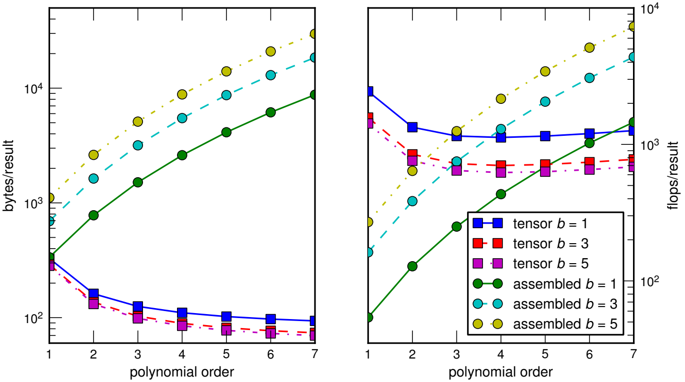



Figure: Scalar Poisson: Expansion factor $nnz(\hat{C})/nnz(A)$, contraction $nnz(\hat{C})/nnz(C)$

- 3D has much higher variability by row
- For elasticity, expansion factor is larger by 3x (for 3D)
- Implementation could batch to limit total memory usage
    - more kernel launches

# Finite element: assembled versus unassembled




- Arithmetic intensity for $Q_p$ elements
  - $< \frac{1}{4}$ (assembled), $\approx 10$ (unassembled), $\approx 4$ to 8 (hardware)
- store Jacobian information at Quass quadrature points
- 70% of peak for $Q_3$ on Nehalem - vectorization within an element
- 30% of peak for $Q_2$ on Sandy Bridge and Haswell - vectorization across elements

# pTatin3d: Lithospheric Dynamics

- Heterogeneous, visco-plastic Stokes with particles for material composition/chemistry, geometric MG with coarse AMG
- May, Brown, Le Pourhiet (SC14)
- Viscous operator application for $Q_2$-$P_1^{\text{disc}}$
- "Tensor": matrix-free implementation using tensor product structure on the reference element
- "Tensor C" absorbs metric term into stored tensor-valued coefficient
- Performance 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 | — | — |

# Cache versus vectorization

- Fundamental trade-off
- Hardware gives us less cache per vector lane
- Intra-element vectorization is complicated and über-custom
- Coordinate transformation is $27 \cdot 9 \cdot \texttt{sizeof(double)} = 1944$ bytes/element.
- Vectorize over 4 or 8 elements, perhaps hardware threads
- L1 cache is not this big: repeated spills in tensor contraction
- This is a *very* simple problem

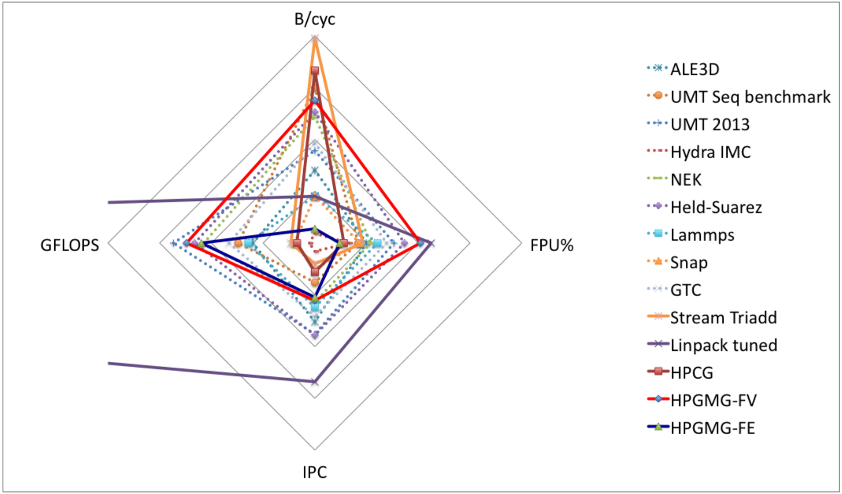
# HPGMG: a new benchmarking proposal

- `https://hpgmg.org`, hpgmg-forum@hpgmg.org mailing list
- SC14 BoF: Wednesday, Nov 19, 12:15pm to 1:15pm
- Mark Adams, Sam Williams (finite-volume), myself (finite-element), John Shalf, Brian Van Straalen, Erich Strohmeier, Rich Vuduc
- Implementations

  Finite Volume memory bandwidth intensive, simple data dependencies

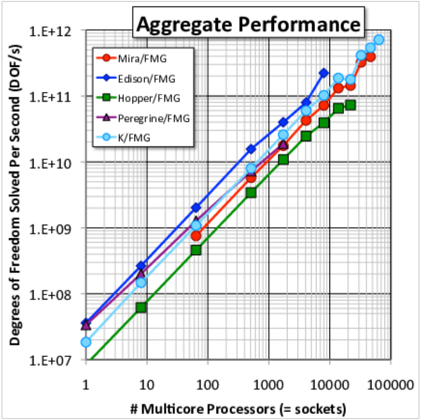
  Finite Element compute- and cache-intensive, vectorizes
- Full multigrid, well-defined, scale-free problem
- Goal: necessary and sufficient
  - Every feature stressed by benchmark should be necessary for an important application
  - Good performance on the benchmark should be sufficient for good performance on most applications

# Kiviat diagrams



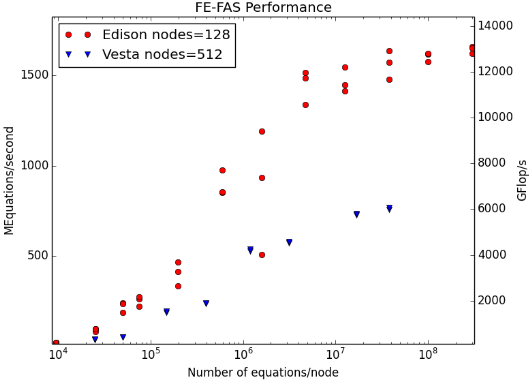

- c/o Ian Karlin and Bert Still (LLNL)

# HPGMG distinguishes networks



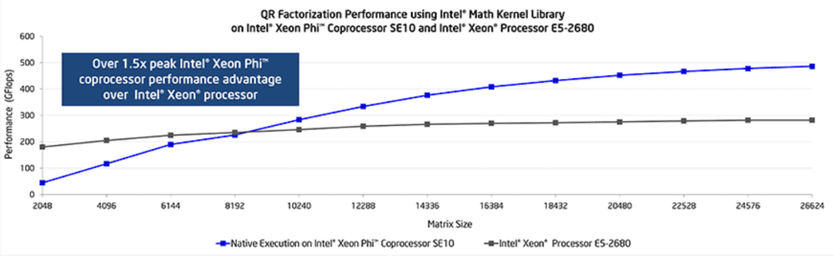

- About 1M dof/socket
- Peregrine and Edison have identical node architecture
- Peregrine has 5:1 tapered IB

# Dynamic Range




- BG/Q vectorization overloads cache, load/store: 88% FXU, 12% FPU
- Users like predictable performance across a range of problem sizes
- Half of all PETSc users care about strong scaling more
- Transient problems do not weak scale even if each step does

# Where we are now: *QR* factorization with MKL on MIC




- Figure compares two CPU sockets (230W TDP) to one MIC (300W TDP plus host)
- Performance/Watt only breaks even at largest problem sizes
- $10^4 \times 10^4$ matrix takes 667 GFlops: about 2 seconds
- This is an $O(n^{3/2})$ operation on $n$ data
- MIC cannot strong scale, no more energy efficient/cost effective

# Outlook

- Memory bandwidth is a major limitation
- Can change algorithms to increase intensity
  - Usually increases stress on cache
- Optimizing for vectorization can incur large bandwidth overhead
- I think data motion is a more fundamental long-term concern
- Latency is at least as important as throughput for many applications
- “hard to program” versus “architecture ill-suited for problem”?
- Performance varies with configuration
  - number of tracers, number of levels, desired steps/second
  - do not need optimality in all cases, but should degrade gracefully