Sparse linear solvers: iterative methods, sparse matrix-vector multiplication, and preconditioning

L. Grigori

ALPINES
INRIA and LJLL, UPMC
On sabbatical at UC Berkeley

March 2015

Plan

Krylov subspace methods
- Conjugate gradient method

Tuning sparse matrix-vector product
- Sequential performance optimization
- Tuning on multicore

Iterative solvers that reduce communication
- CA solvers based on s-step methods
- Enlarged Krylov methods

Preconditioners
- One level preconditioners: CA-ILU0
- Two level preconditioners

Extra slides: one level preconditioners
- One level preconditioners: examples

Plan

Krylov subspace methods
    Conjugate gradient method

Tuning sparse matrix-vector product

Iterative solvers that reduce communication

Preconditioners

Extra slides: one level preconditioners

Krylov subspace methods

Solve $Ax = b$ by finding a sequence $x_1, x_2, ..., x_k$ that minimizes some measure of error over the corresponding spaces

$$x_0 + \mathcal{K}_i(A, r_0), \quad i = 1, ..., k$$

.

They are defined by two conditions:

1. Subspace condition: $x_k \in x_0 + \mathcal{K}_k(A, r_0)$
2. Petrov-Galerkin condition: $r_k \perp \mathscr{L}_k$

$$\iff (r_k)^t y = 0, \quad \forall \quad y \in \mathscr{L}_k$$

where

- x_0 is the initial iterate, r_0 is the initial residual,
- $\mathcal{K}_k(A, r_0) = span\{r_0, Ar_0, A^2 r_0, ..., A^{k-1} r_0\}$ is the Krylov subspace of dimension k,
- $\mathscr{L}_k$ is a well-defined subspace of dimension k.

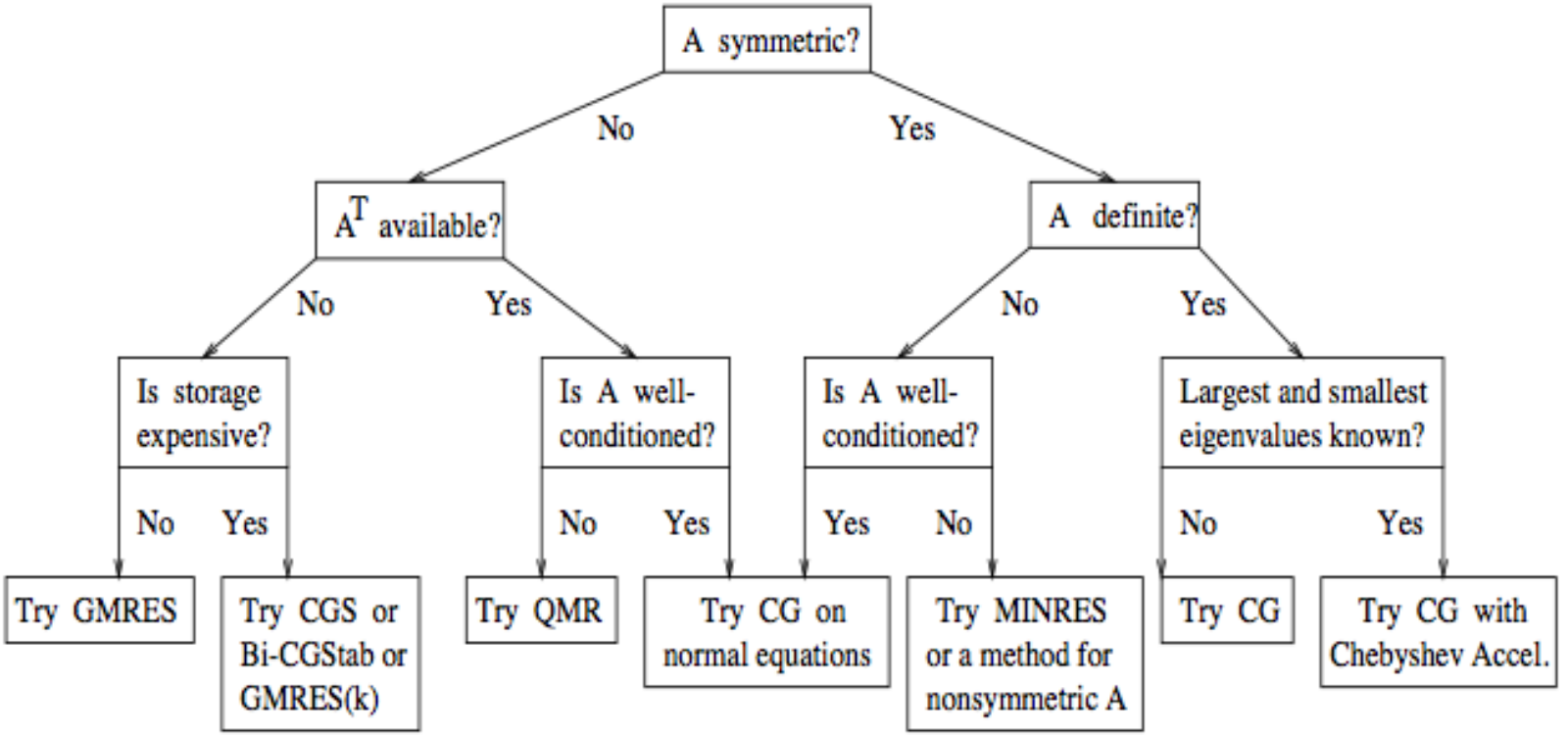
One of Top Ten Algorithms of the 20th Century

From SIAM News, Volume 33, Number 4:
Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of Krylov subspace iteration methods.

- Russian mathematician Alexei Krylov writes first paper, 1931.
- Lanczos - introduced an algorithm to generate an orthogonal basis for such a subspace when the matrix is symmetric.
- Hestenes and Stiefel - introduced CG for SPD matrices.

Other Top Ten Algorithms: Monte Carlo method, decompositional approach to matrix computations (Householder), Quicksort, Fast multipole, FFT.

Choosing a Krylov method



All methods (GMRES, CGS,CG…) depend on SpMV (or variations…)

See www.netlib.org/templates/Templates.html for details

Source slide: J. Demmel

Conjugate gradient (Hestenes, Stieffel, 52)

- A Krylov projection method for SPD matrices where $\mathscr{L}_k = \mathcal{K}_k(A, r_0)$.
- Finds $x^* = A^{-1}b$ by minimizing the quadratic function

$$\begin{aligned} \phi(x) &= \frac{1}{2}(x)^t Ax - b^t x \\ \bigtriangledown\phi(x) &= Ax - b = 0 \end{aligned}$$

- After j iterations of CG,

$$||x^* - x_j||_A \leq 2||x - x_0||_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^j,$$

where x_0 is starting vector, $||x||_A = \sqrt{x^T Ax}$ and $\kappa(A) = |\lambda_{max}(A)|/|\lambda_{min}(A)|$.

Conjugate gradient

- Computes A-orthogonal search directions by conjugation of the residuals

$$\left\{ \begin{array}{lcl} p_1 & = & r_0 = - \bigtriangledown \phi(x_0) \\ p_k & = & r_{k-1} + \beta_k p_{k-1} \end{array} \right. \tag{1}$$

- At k-th iteration,

$$x_k = x_{k-1} + \alpha_k p_k = argmin_{x \in x_0 + \mathcal{K}_k(A, r_0)} \phi(x)$$

where α_k is the step along p_k.

- CG algorithm obtained by imposing the orthogonality and the conjugacy conditions

$$\begin{array}{rcl} r_k^T r_i & = & 0, \text{for all } i \neq k, \\ p_k^T A p_i & = & 0, \text{for all } i \neq k. \end{array}$$

CG algorithm

Algorithm 1 The CG Algorithm

1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2^2$, $p_1 = r_0$, $k = 1$
2: **while** ($\sqrt{\rho_k} > \epsilon||b||_2$ and $k < k_{max}$) **do**
3: **if** $(k \neq 1)$ **then**
4: $\beta_k = (r_{k-1}, r_{k-1})/(r_{k-2}, r_{k-2})$
5: $p_k = r_{k-1} + \beta_k p_{k-1}$
6: **end if**
7: $\alpha_k = (r_{k-1}, r_{k-1})/(Ap_k, p_k)$
8: $x_k = x_{k-1} + \alpha_k p_k$
9: $r_k = r_{k-1} - \alpha_k Ap_k$
10: $\rho_k = ||r_k||_2^2$
11: $k = k + 1$
12: **end while**

Challenge in getting efficient and scalable solvers

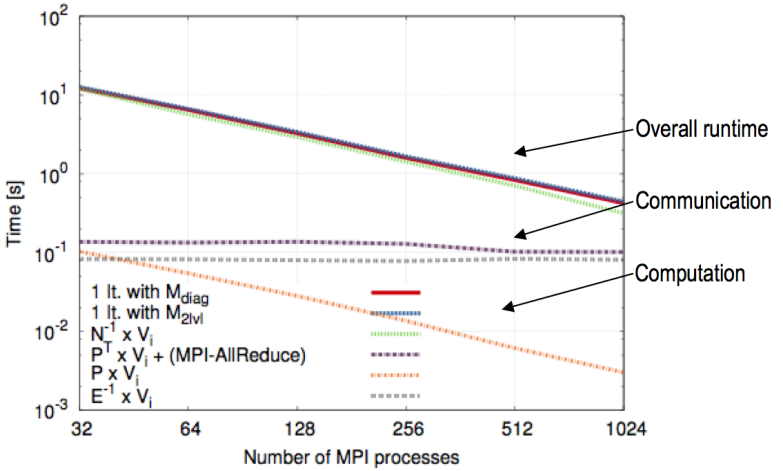
- A Krylov solver finds x_{k+1} from $x_0 + \mathcal{K}_{k+1}(A, r_0)$ where

$$\mathcal{K}_{k+1}(A, r_0) = span\{r_0, Ar_0, A^2 r_0, ..., A^k r_0\},$$

  such that the Petrov-Galerkin condition $b - Ax_{k+1} \perp \mathscr{L}_{k+1}$ is satisfied.
- Does a sequence of k SpMVs to get vectors $[x_1, ..., x_k]$
- Finds best solution x_{k+1} as linear combination of $[x_1, ..., x_k]$

Typically, each iteration requires

- Sparse matrix vector product
  $\rightarrow$ point-to-point communication
- Dot products for orthogonalization
  $\rightarrow$ global communication



Map making, with R. Stompor, M. Szydlarski
Results obtained on Hopper, Cray XE6, NERSC

Challenge in getting efficient and scalable solvers

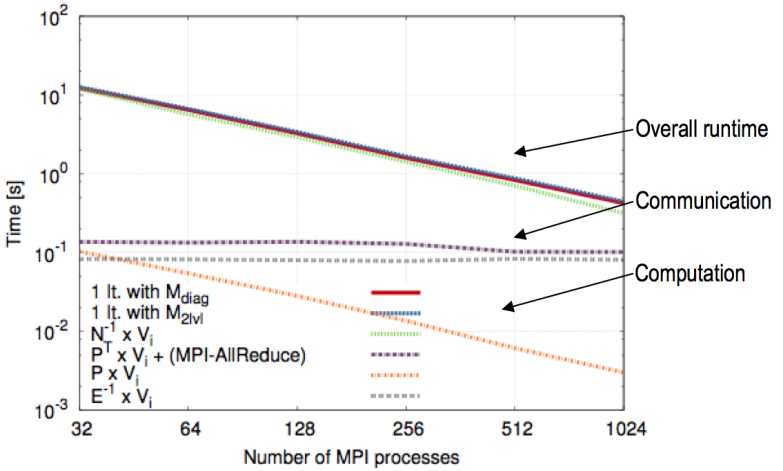
- A Krylov solver finds x_{k+1} from $x_0 + \mathcal{K}_{k+1}(A, r_0)$ where

$$\mathcal{K}_{k+1}(A, r_0) = span\{r_0, Ar_0, A^2 r_0, ..., A^k r_0\},$$

  such that the Petrov-Galerkin condition $b - Ax_{k+1} \perp \mathscr{L}_{k+1}$ is satisfied.
- Does a sequence of k SpMVs to get vectors $[x_1, ..., x_k]$
- Finds best solution x_{k+1} as linear combination of $[x_1, ..., x_k]$

Typically, each iteration requires

- Sparse matrix vector product
  $\rightarrow$ point-to-point communication
- Dot products for orthogonalization
  $\rightarrow$ global communication



Map making, with R. Stompor, M. Szydlarski
Results obtained on Hopper, Cray XE6, NERSC

Ways to improve performance

We will look at three different approaches:

- Improve the performance of sparse matrix-vector product.
- Change numerics - reformulate or introduce Krylov subspace algorithms to:
  - reduce communication,
  - increase arithmetic intensity - compute sparse matrix-set of vectors product.
- Use preconditioners to decrease the number of iterations till convergence.

Plan

Krylov subspace methods

Tuning sparse matrix-vector product
- Sequential performance optimization
- Tuning on multicore

Iterative solvers that reduce communication

Preconditioners

Extra slides: one level preconditioners

Tuning sparse matrix-vector product

- Slides from J. Demmel, lecture on *Automatic Performance Tuning and Sparse-Matrix-Vector-Multiplication (SpMV)*
  `www.cs.berkeley.edu/~demmel/cs267_Spr14`
- Sequential performance optimization
- Tuning SpMV on multicores
- Most of the techniques discussed are available in **OSKI** and **pOSKI**: Optimized Sparse Kernel Interface
  `bebop.cs.berkeley.edu/poski`
  - Provides sparse kernels automatically tuned for user's matrix & machine.

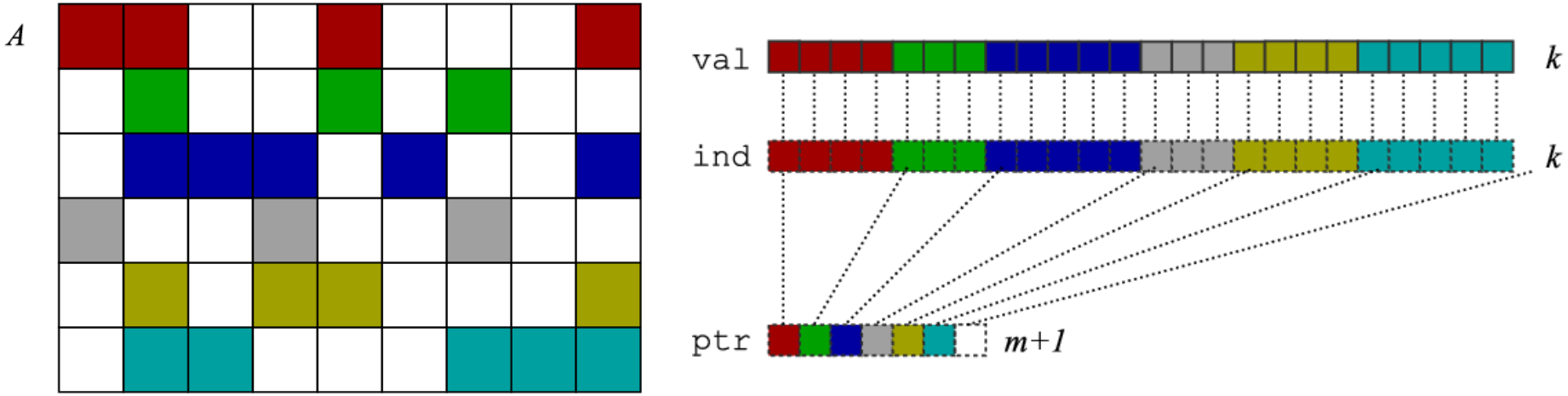
Examples of Automatic Performance Tuning (1)

- Dense BLAS (PHiPAC-UCB, then ATLAS-UTK), FFTs (FFTw – MIT), signal processing(SPIRAL - CMU), MPI reductions
- What do they have in common?
  - Can do the tuning **off-line**: once per architecture, algorithm
  - Can take as much time as necessary (hours, a week…)
  - At run-time, algorithm choice may depend only on few parameters
    - Matrix dimension, size of FFT, etc.

Examples of Automatic Performance Tuning (2)

- What do dense BLAS, FFTs, signal processing, MPI reductions have in common?
  - Can do the tuning **off-line**: once per architecture, algorithm
  - Can take as much time as necessary (hours, a week…)
  - At run-time, algorithm choice may depend only on few parameters
    - Matrix dimension, size of FFT, etc.
- **Can't always do off-line tuning**
  - **Algorithm and implementation may strongly depend on data only known at run-time**
  - **Ex: Sparse matrix nonzero pattern determines both best data structure and implementation of Sparse-matrix-vector-multiplication (SpMV)**
  - **Part of search for best algorithm just be done (very quickly!) at run-time**

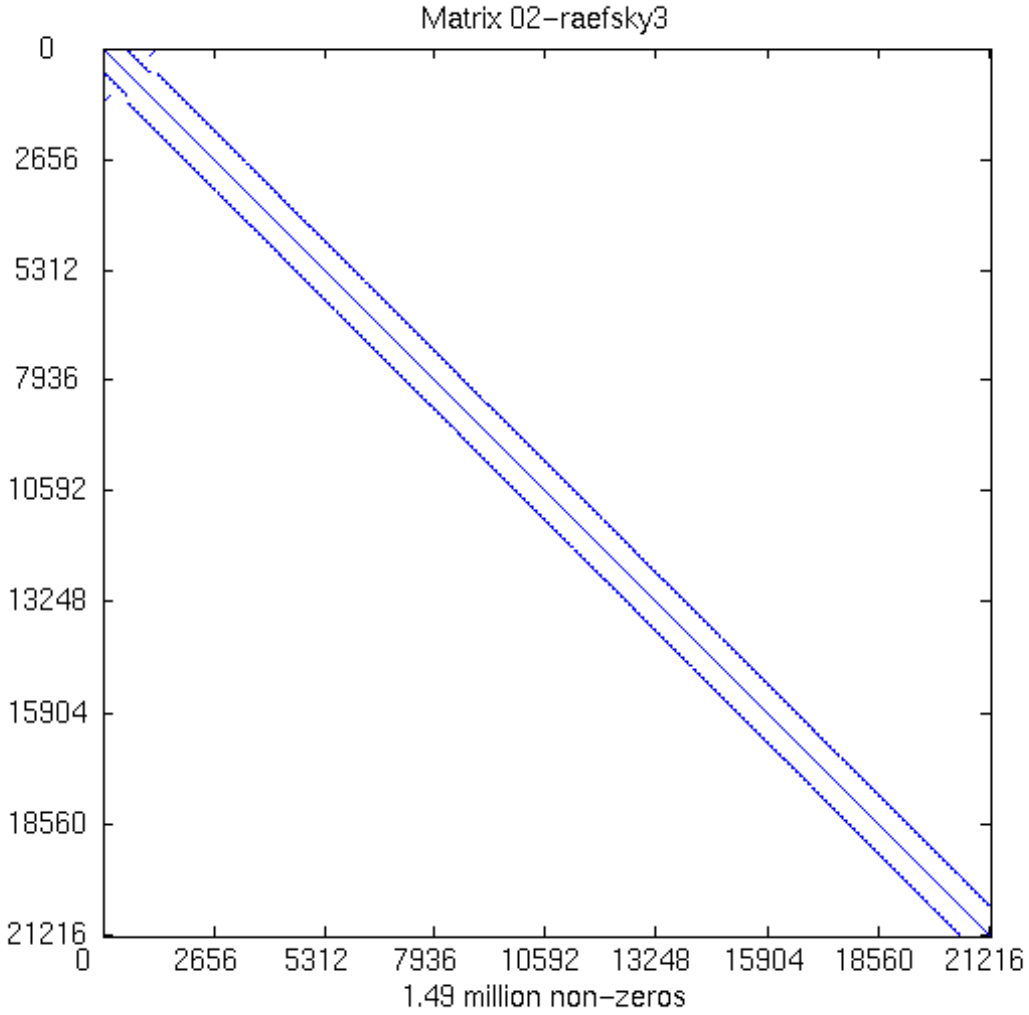
SpMV with Compressed Sparse Row (CSR) Storage



Matrix-vector multiply kernel: y(i) ← y(i) + A(i,j)*x(j)

```
for each row i
   for k=ptr[i] to ptr[i+1]-1 do
       y[i] = y[i] + val[k]*x[ind[k]]
```

Example: The Difficulty of Tuning



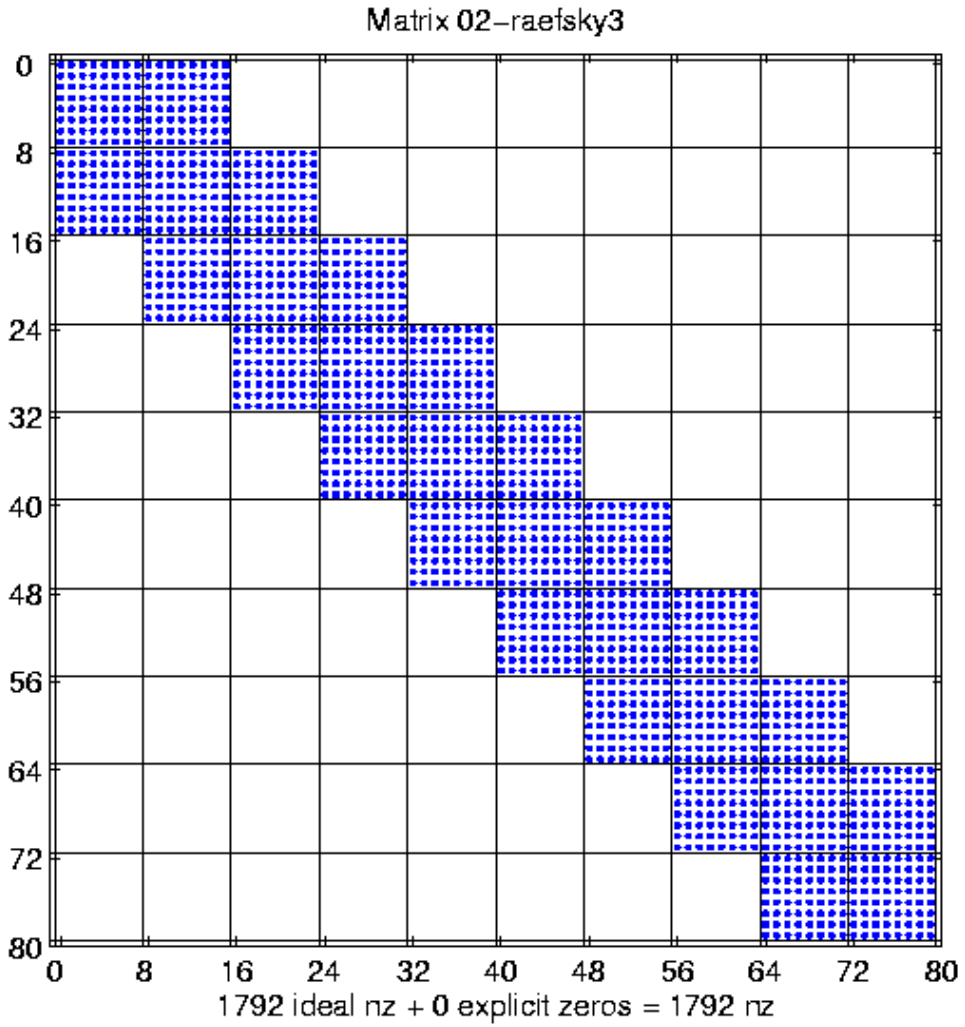
- n = 21200
- nnz = 1.5 M
- kernel: SpMV

- Source:

  FEM discretization

  NASA structural analysis problem

Example: The Difficulty of Tuning



- n = 21200
- nnz = 1.5 M
- kernel: SpMV

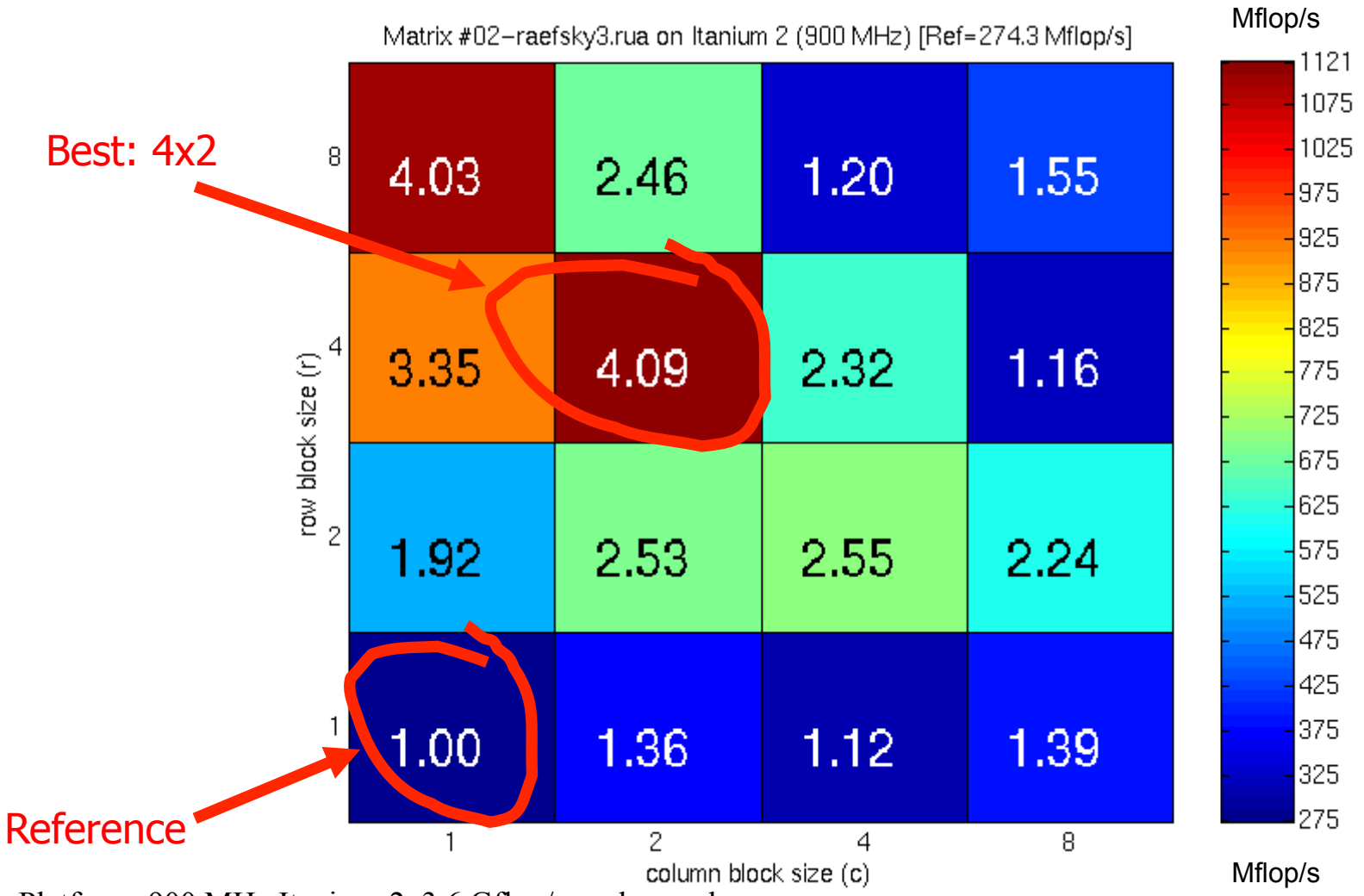
- Source: NASA structural analysis problem

- **8x8** dense substructure

Taking advantage of block structure in SpMV

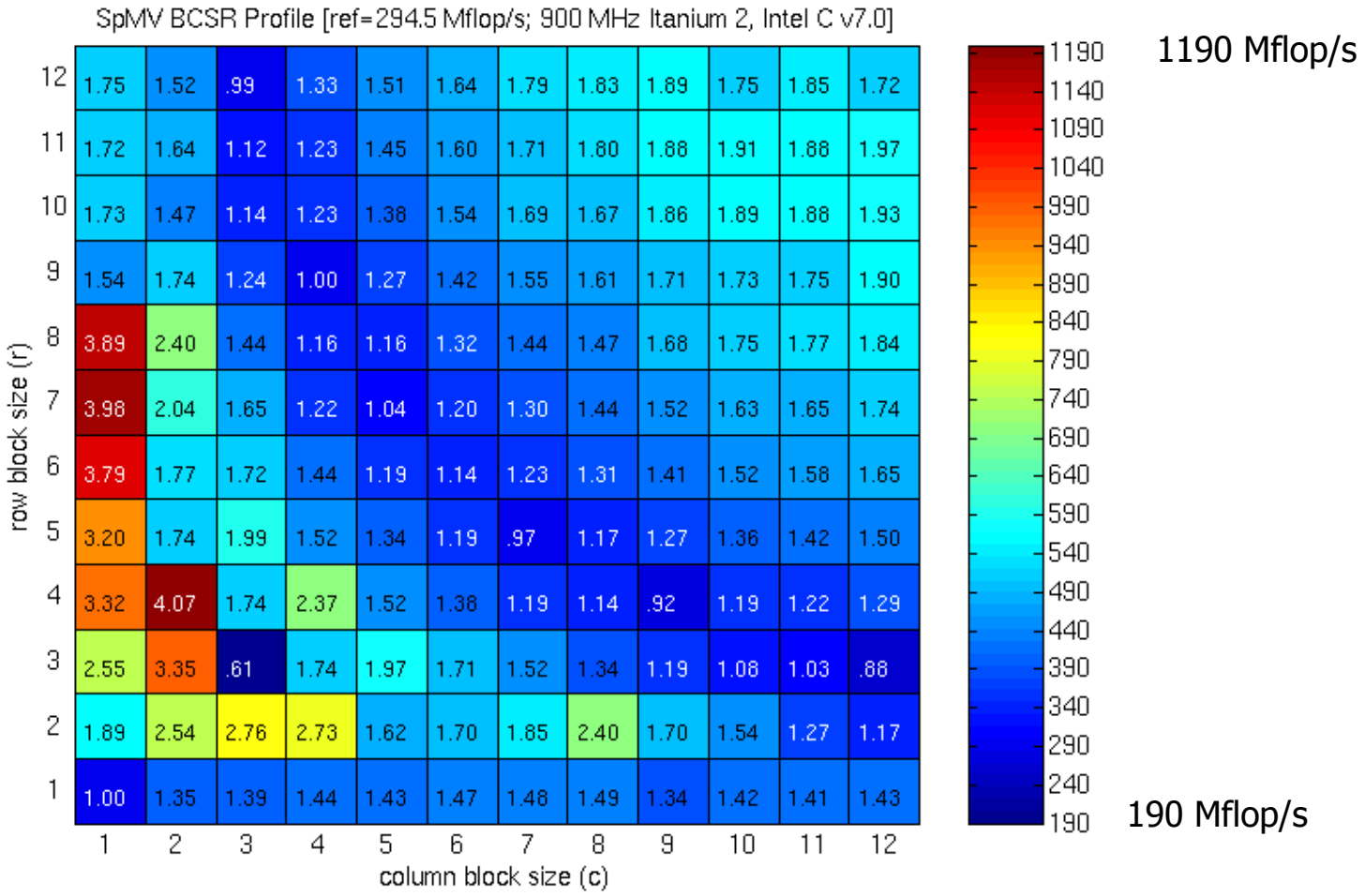
- Bottleneck is time to get matrix from memory
  - Only 2 flops for each nonzero in matrix
- Don't store each nonzero with index, instead store each nonzero r-by-c block with index
  - Storage drops by up to 2x, if rc >> 1, all 32-bit quantities
  - Time to fetch matrix from memory decreases
- Change both data structure and algorithm
  - Need to pick r and c
  - Need to change algorithm accordingly
- In example, is r=c=8 best choice?
  - Minimizes storage, so looks like a good idea…

Speedups on Itanium 2: The Need for Search



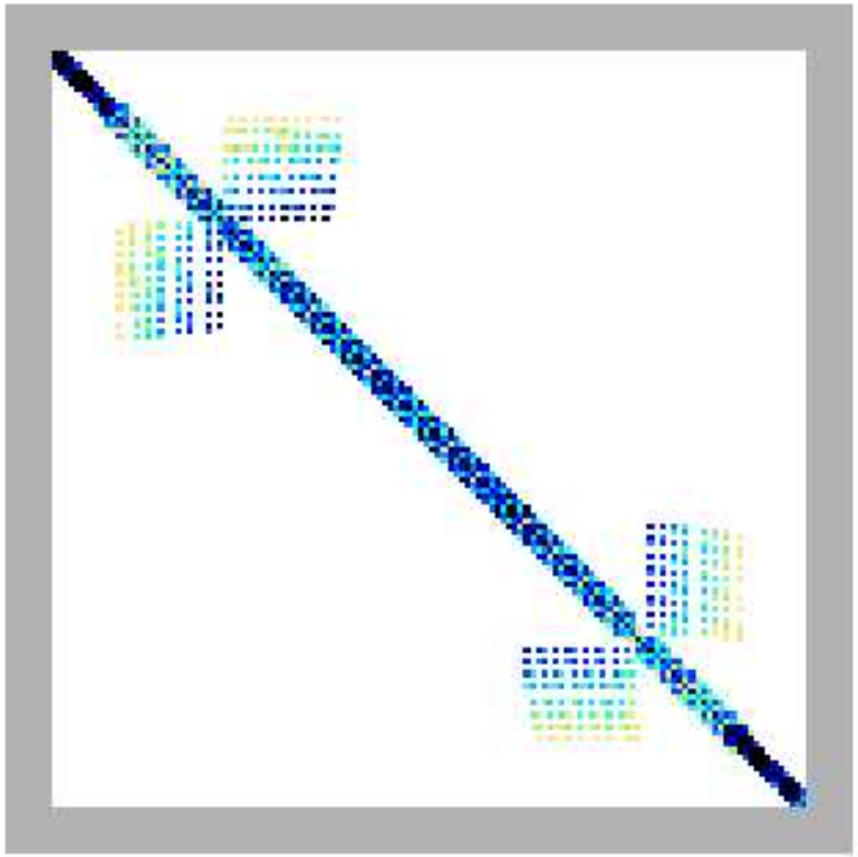
Platform: 900 MHz Itanium-2, 3.6 Gflop/s peak speed.

Register Profile: Itanium 2



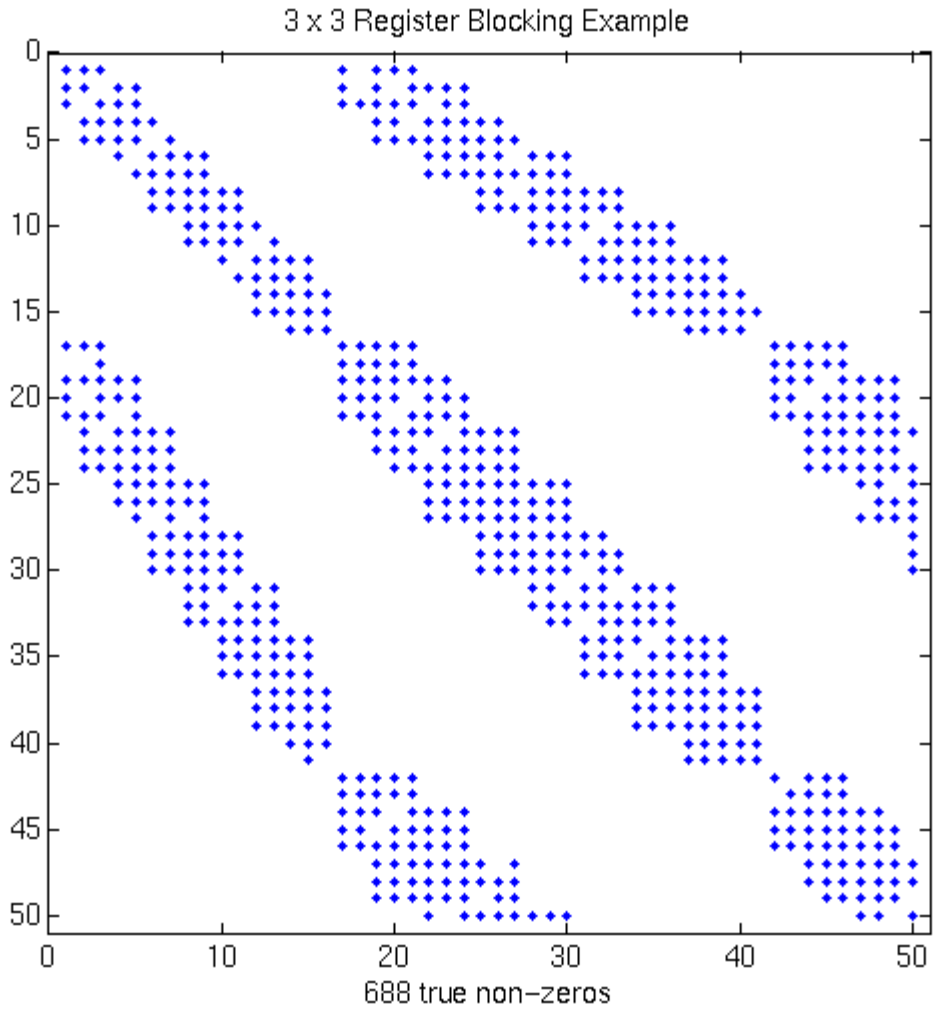
Example of off-line tuning: dense matrix

Another example of tuning challenges



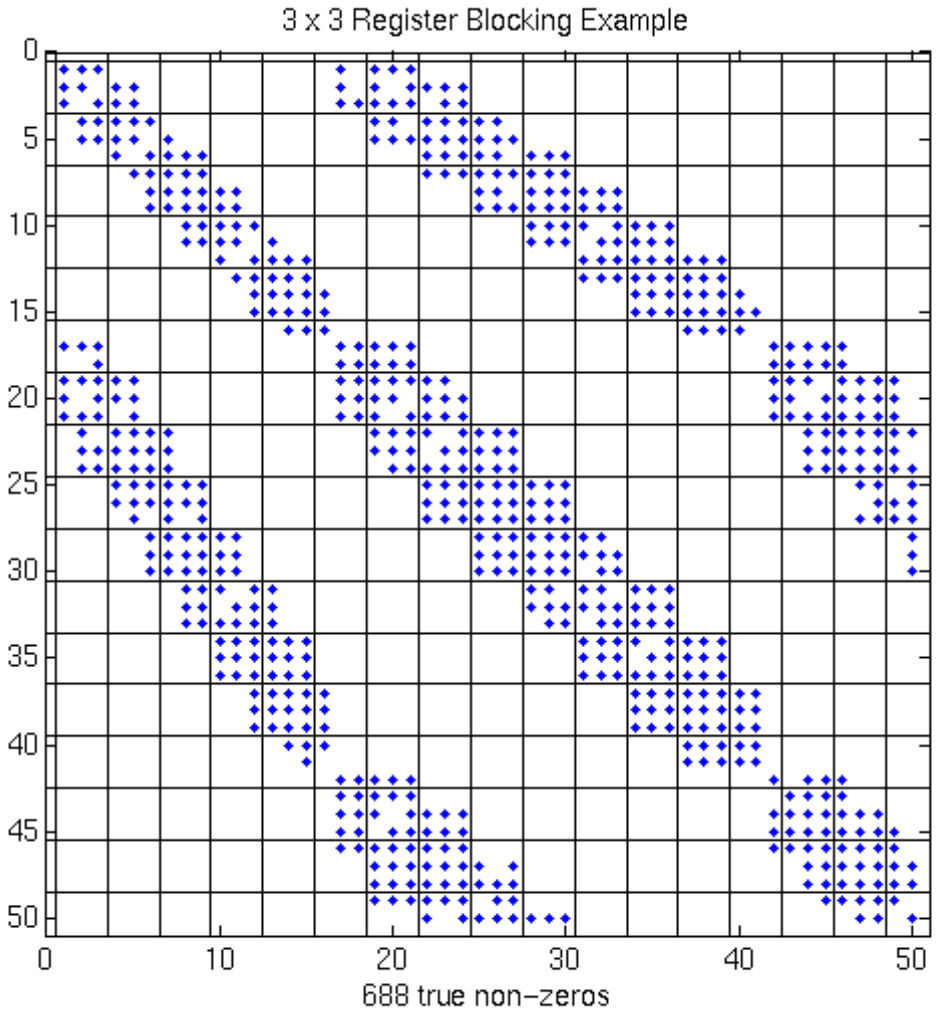
- More complicated non-zero structure in general
- N = 16614
- NNZ = 1.1M
- FEM fluid flow application

Zoom in to top corner



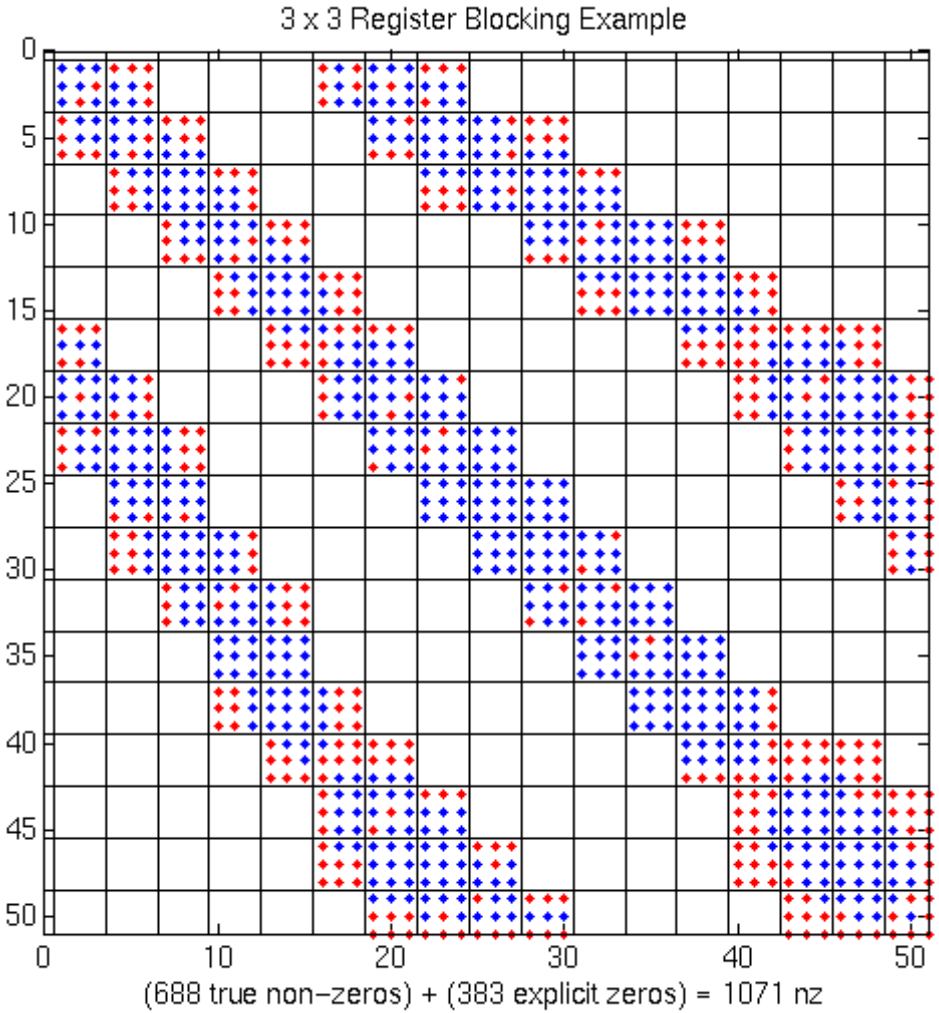
- More complicated non-zero structure in general
- N = 16614
- NNZ = 1.1M

3x3 blocks look natural, but…



- More complicated non-zero structure in general
- Example: 3x3 blocking
  - Logical grid of 3x3 cells
- But would lead to lots of “fill-in”

Extra Work Can Improve Efficiency!



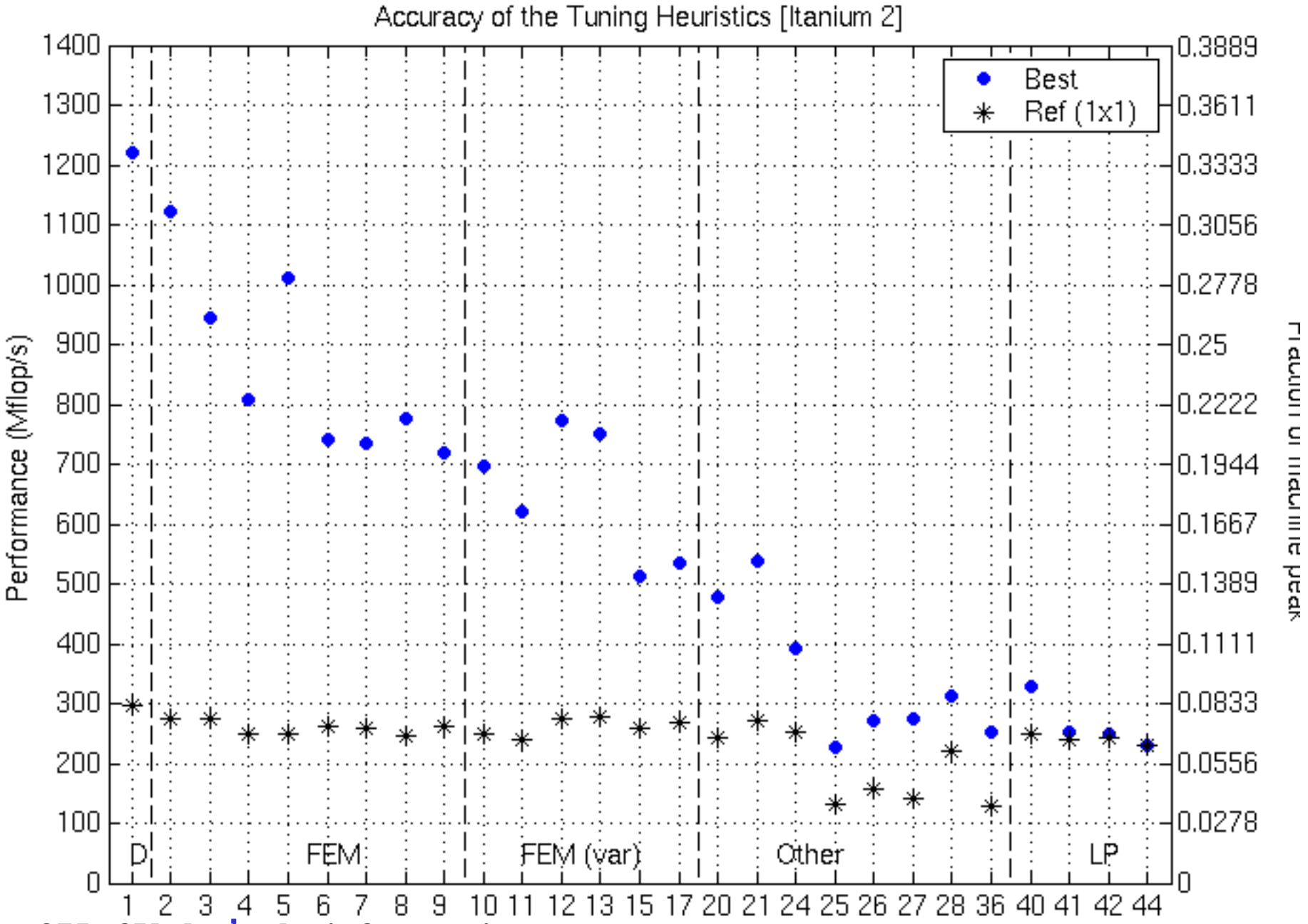
- More complicated non-zero structure in general
- Example: 3x3 blocking
  - Logical grid of 3x3 cells
  - Fill-in explicit zeros
  - Unroll 3x3 block multiplies
  - "Fill ratio" = 1.5
- On Pentium III: 1.5x speedup!
  - Actual mflop rate $1.5^2 = 2.25$ higher

Automatic Register Block Size Selection

- Selecting the r x c block size
  - **Off-line benchmark**
    - Precompute **Mflops(r,c)** using dense A for each r x c
    - Once per machine/architecture
  - **Run-time "search"**
    - Sample *A* to estimate **Fill(r,c)** for each r x c
  - **Run-time heuristic model**
    - Choose r, c to minimize **time ~ Fill(r,c) / Mflops(r,c)**

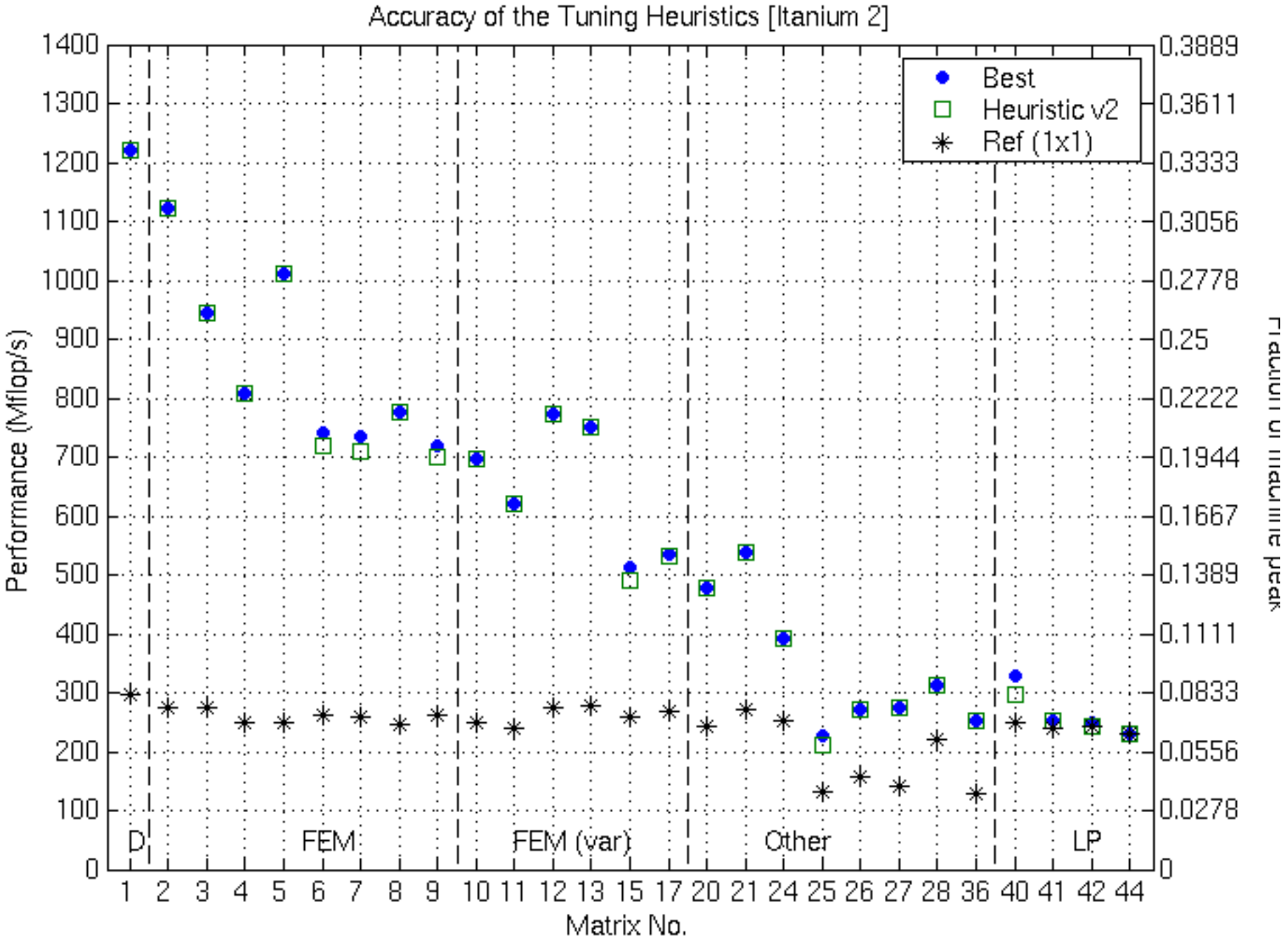
Accurate and Efficient Adaptive Fill Estimation

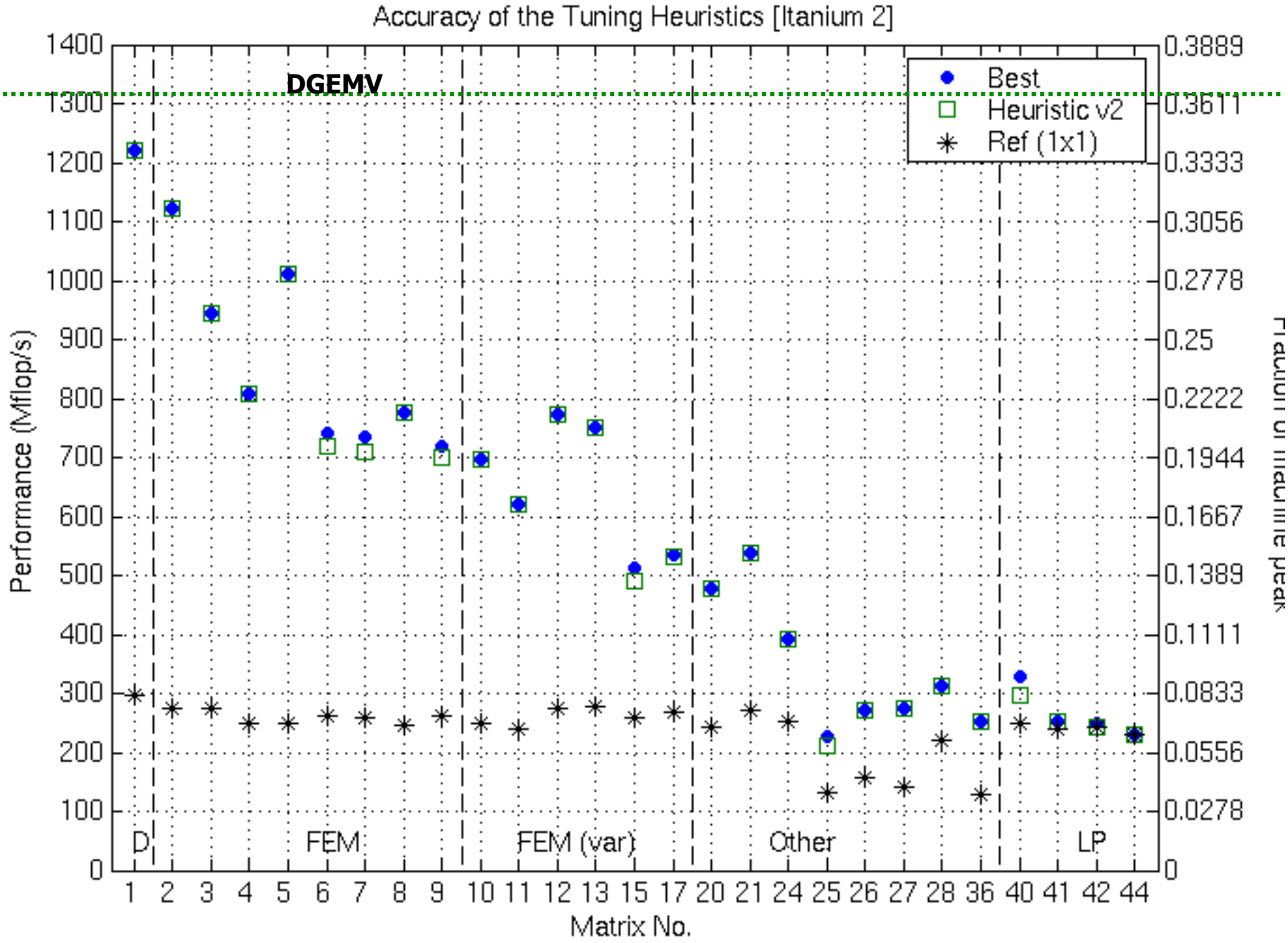
- Idea: Sample matrix
  - Fraction of matrix to sample: $s \in [0,1]$
  - Cost ~ O(s * nnz)
  - Control cost by controlling s
    - Search at run-time: the constant matters!
- Control s automatically by computing statistical confidence intervals
  - Idea: Monitor variance
- Cost of tuning
  - Lower bound: convert matrix in 5 to 40 unblocked SpMVs
  - Heuristic: 1 to 11 SpMVs



See p. 375 of Vuduc's thesis for matrices

NOTE: "Fair" flops used (ops on explicit zeros not counted as "work")


Accuracy of the Tuning Heuristics [Itanium 2]
Performance (Mflop/s)
Fraction of Machine Peak
Matrix No.
Best
Heuristic v2
Ref (1x1)
D
FEM
FEM (var)
Other
LP
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
0.3889
0.3611
0.3333
0.3056
0.2778
0.25
0.2222
0.1944
0.1667
0.1389
0.1111
0.0833
0.0556
0.0278
0
1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 20 21 24 25 26 27 28 36 40 41 42 44


Accuracy of the Tuning Heuristics [Itanium 2]
DGEMV
Best
Heuristic v2
Ref (1x1)
Performance (Mflop/s)
Fraction of Machine Peak
Matrix No.
D
FEM
FEM (var)
Other
LP
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
0.0278
0.0556
0.0833
0.1111
0.1389
0.1667
0.1944
0.2222
0.25
0.2778
0.3056
0.3333
0.3611
0.3889
1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 20 21 24 25 26 27 28 36 40 41 42 44

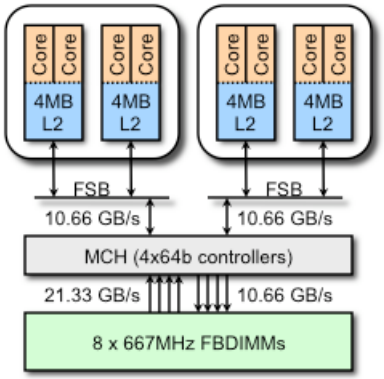
Summary of Other Sequential Performance Optimizations

- Optimizations for SpMV
  - **Register blocking (RB)**: up to **4x** over CSR
  - **Variable block splitting**: **2.1x** over CSR, 1.8x over RB
  - **Diagonals**: **2x** over CSR
  - **Reordering** to create dense structure + **splitting**: **2x** over CSR
  - **Symmetry**: **2.8x** over CSR, 2.6x over RB
  - **Cache blocking**: **2.8x** over CSR
  - **Multiple vectors (SpMM)**: **7x** over CSR
  - And combinations…
- Sparse triangular solve
  - Hybrid sparse/dense data structure: **1.8x** over CSR
- Higher-level kernels
  - $\mathbf{A \cdot A^T \cdot x}$, $\mathbf{A^T \cdot A \cdot x}$: **4x** over CSR, 1.8x over RB
  - $\mathbf{A^2 \cdot x}$: **2x** over CSR, 1.5x over RB
  - $[A \cdot x, A^2 \cdot x, A^3 \cdot x, .. , A^k \cdot x]$

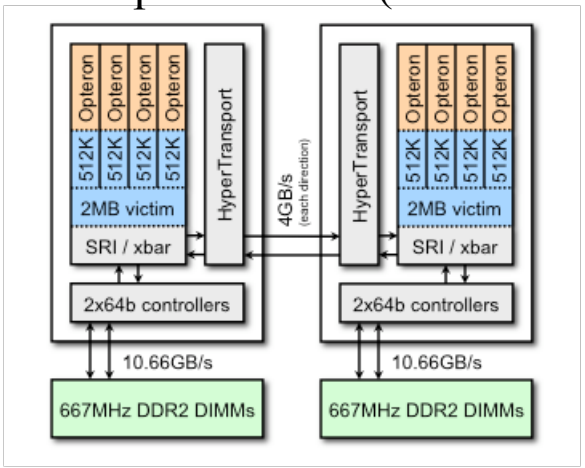
Tuning SpMV on Multicore

Multicore SMPs Used

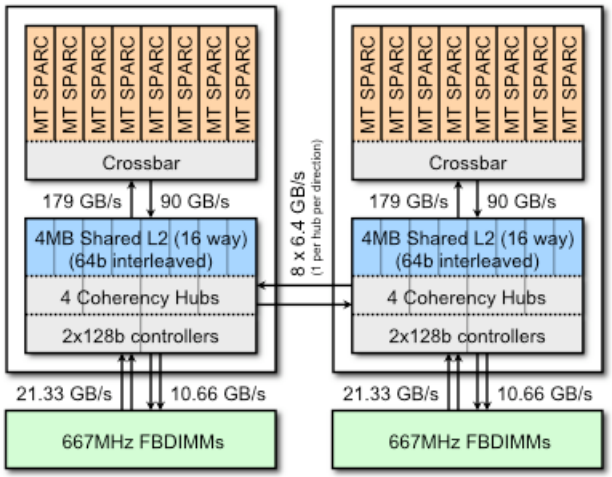
Intel Xeon E5345 (Clovertown)



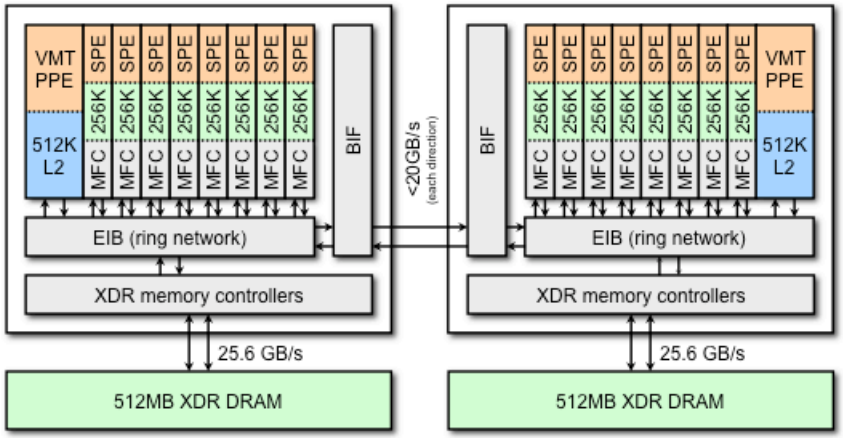
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

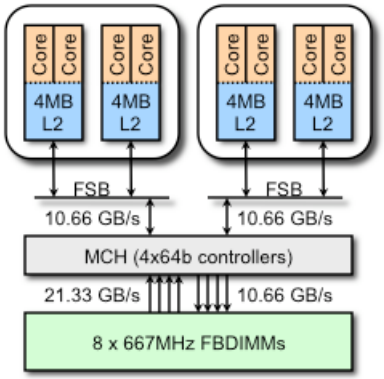


 Source: Sam Williams

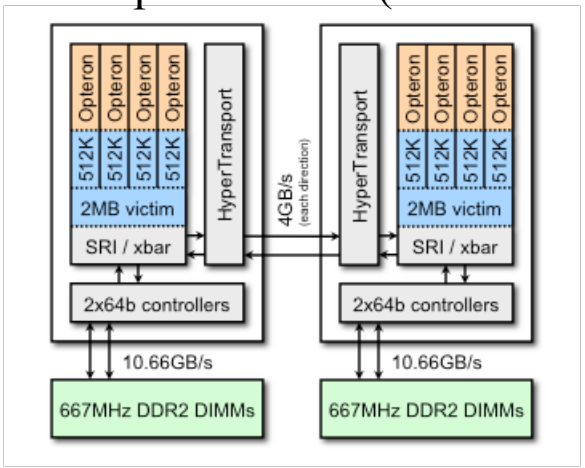
Multicore SMPs Used

(Conventional cache-based memory hierarchy)

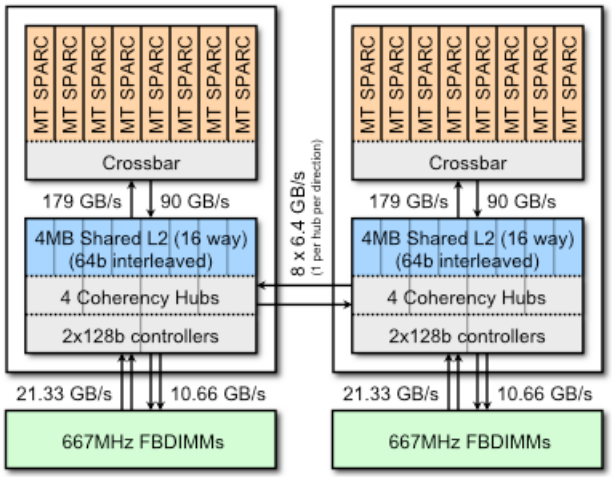
Intel Xeon E5345 (Clovertown)



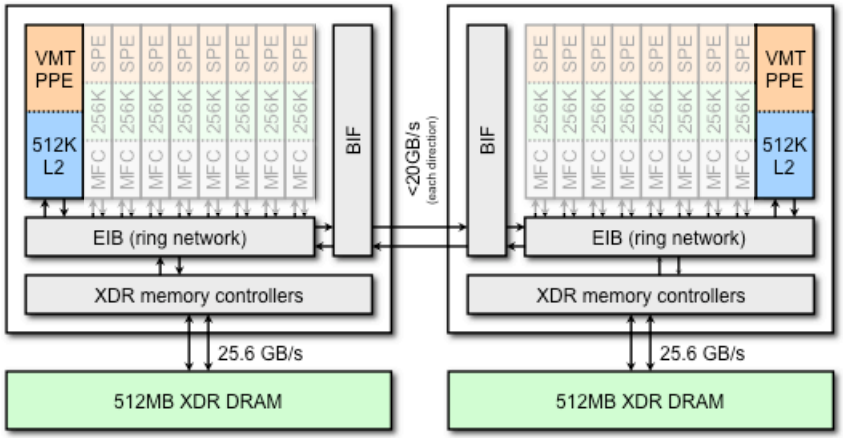
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

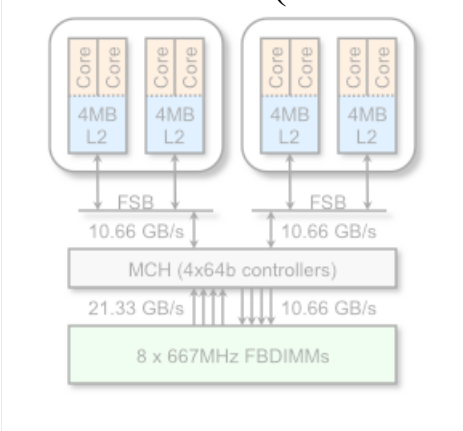


2 such Control processors PPEs on Cell

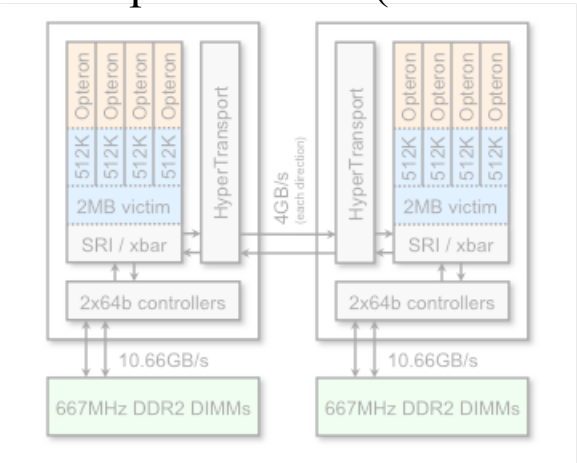
 Source: Sam Williams

Multicore SMPs Used
(Local store-based memory hierarchy)

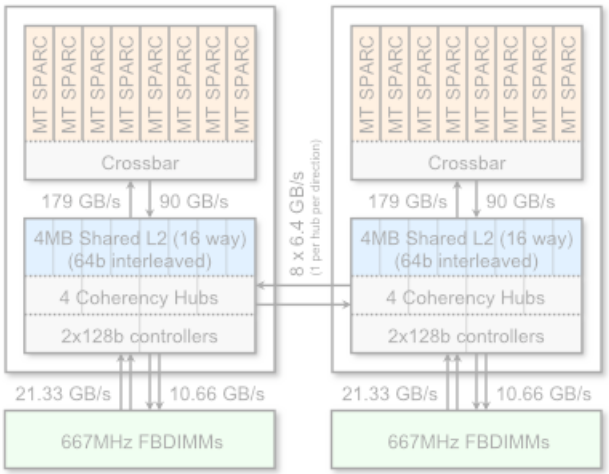
Intel Xeon E5345 (Clovertown)



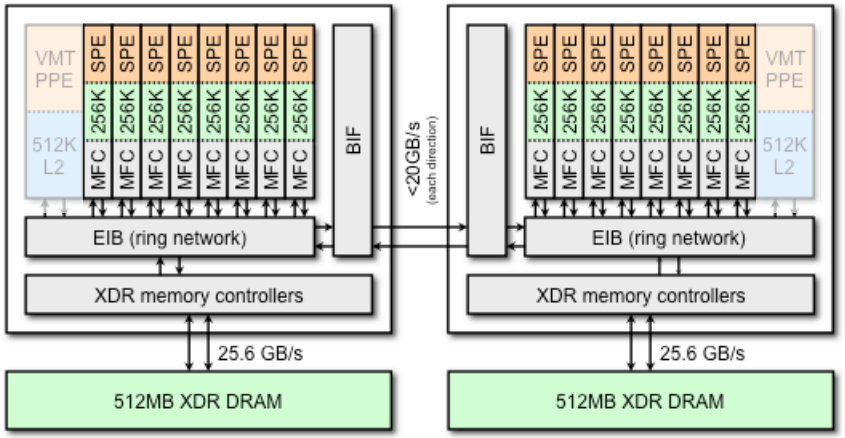
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

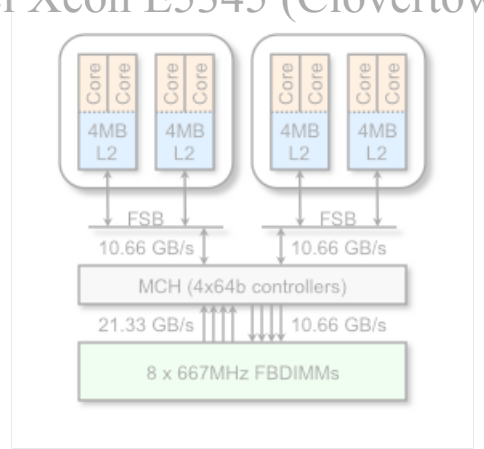


Explicit load and stores (special subroutines) for 16 SPEs
To move data between local memory/DRAM

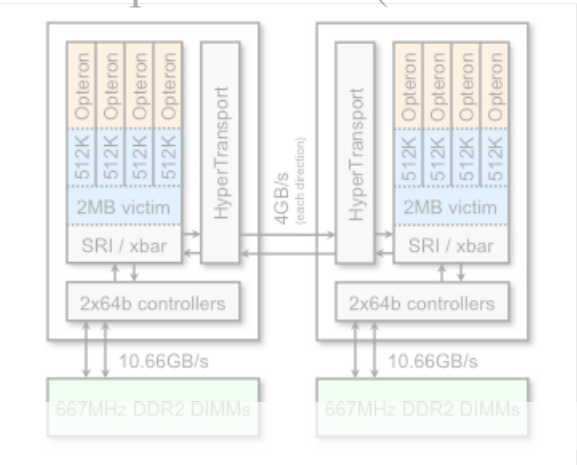
Source: Sam Williams

Multicore SMPs Used
(CMT = Chip-MultiThreading)

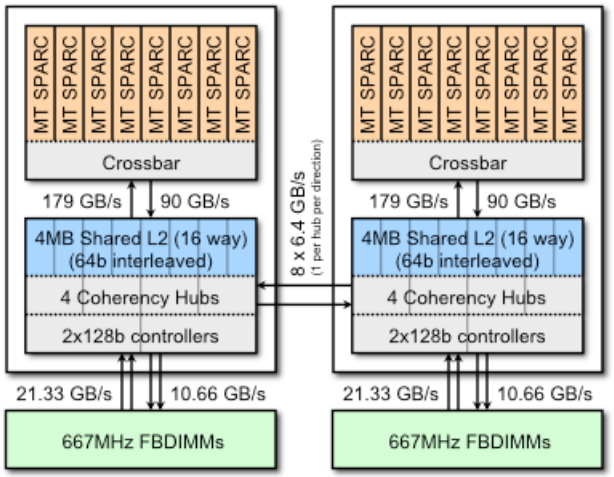
Intel Xeon E5345 (Clovertown)



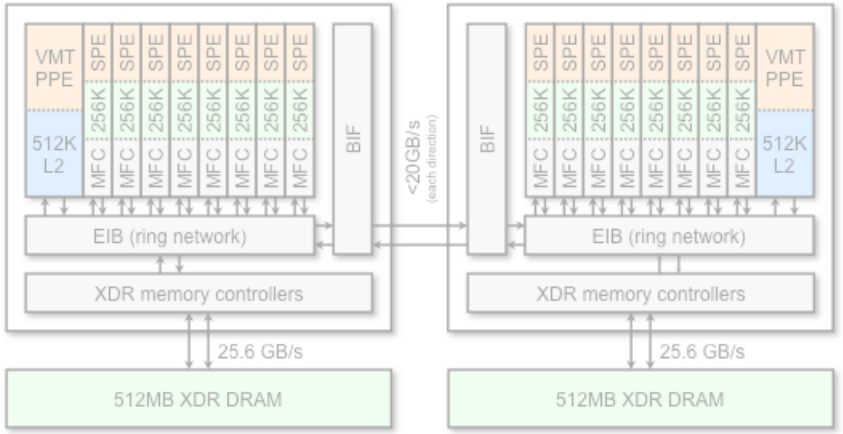
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

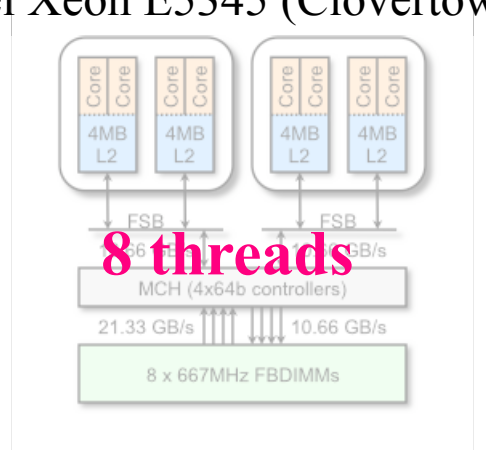


HW switches automatically from thread waiting for memory to another

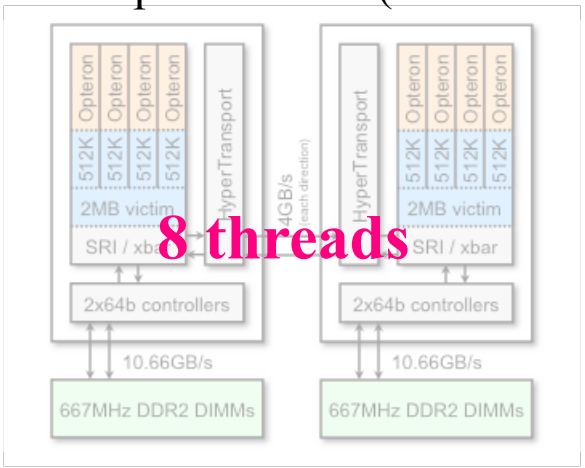
 Source: Sam Williams

Multicore SMPs Used
(threads)

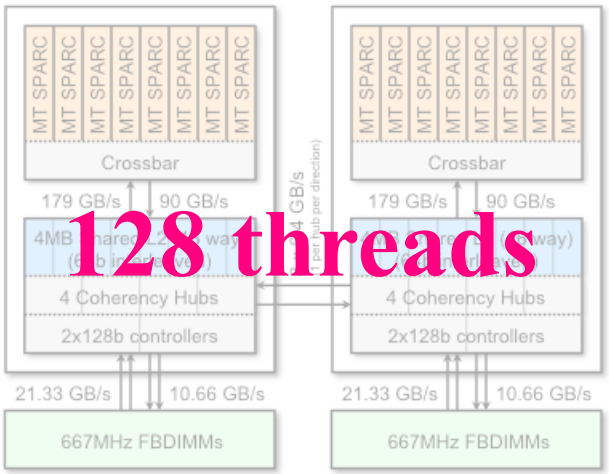
Intel Xeon E5345 (Clovertown)



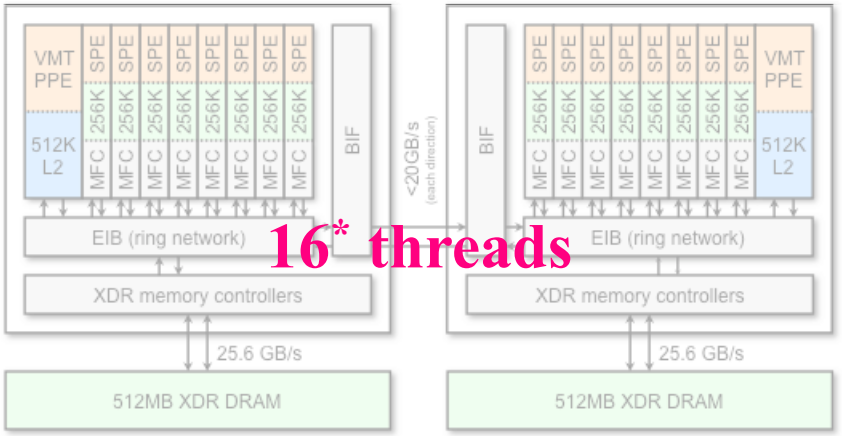
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

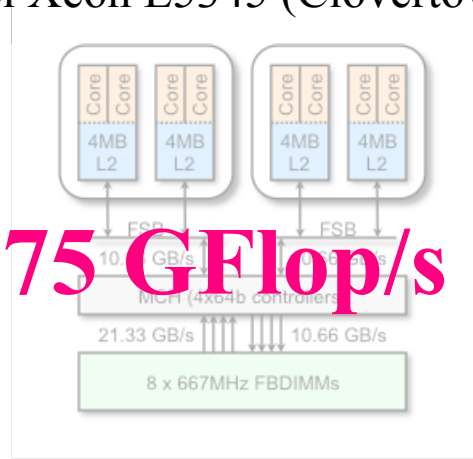


Source: Sam Williams

*SPEs only

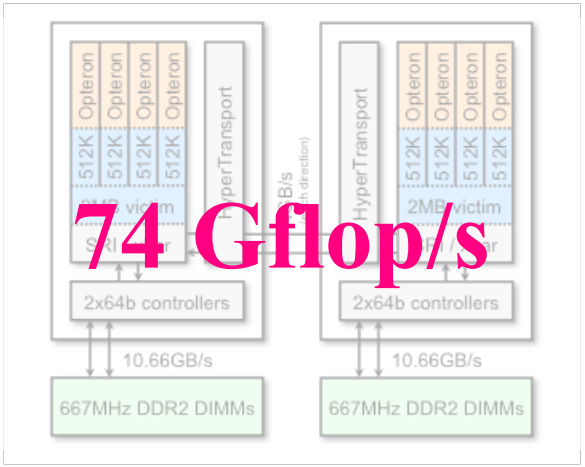
Multicore SMPs Used
(peak double precision flops)

Intel Xeon E5345 (Clovertown)



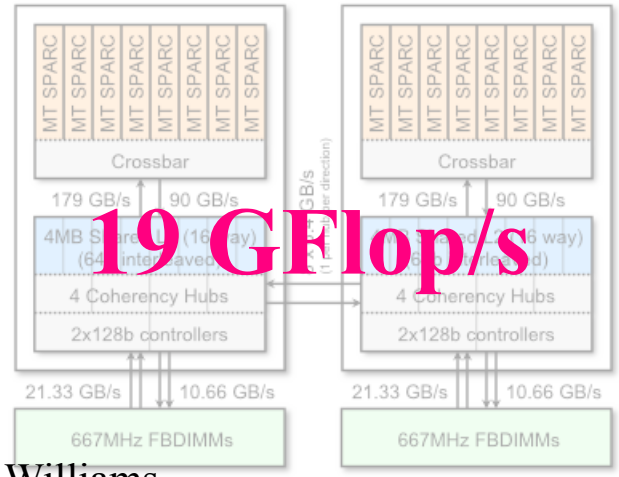
75 GFlop/s

AMD Opteron 2356 (Barcelona)



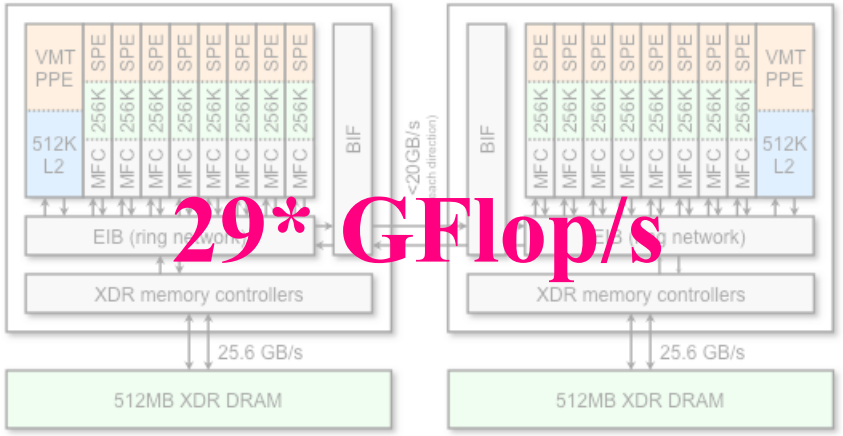
74 Gflop/s

Sun T2+ T5140 (Victoria Falls)



19 GFlop/s

IBM QS20 Cell Blade



29* GFlop/s

Source: Sam Williams

*SPEs only

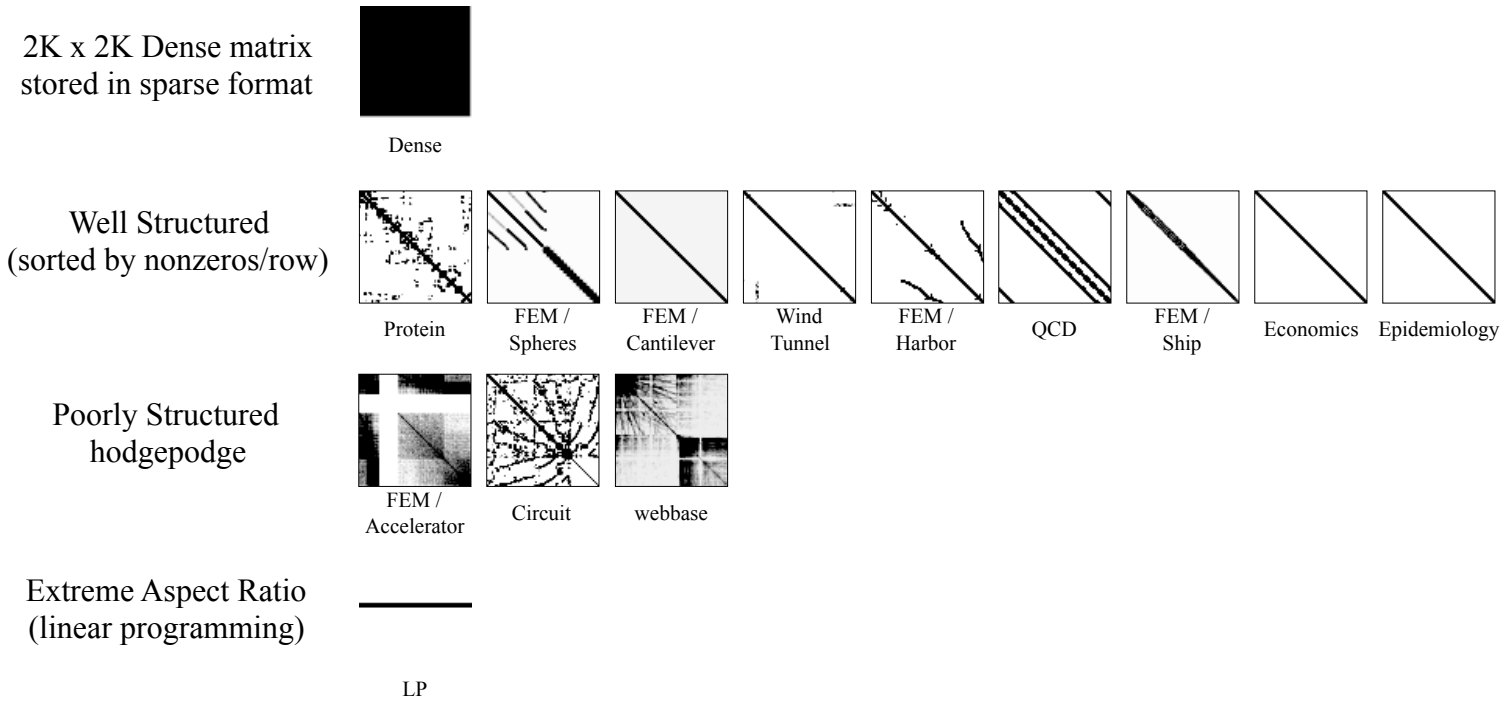
Results from

"Auto-tuning Sparse Matrix-Vector Multiplication (SpMV)"

Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", Supercomputing (SC), 2007.

Test matrices

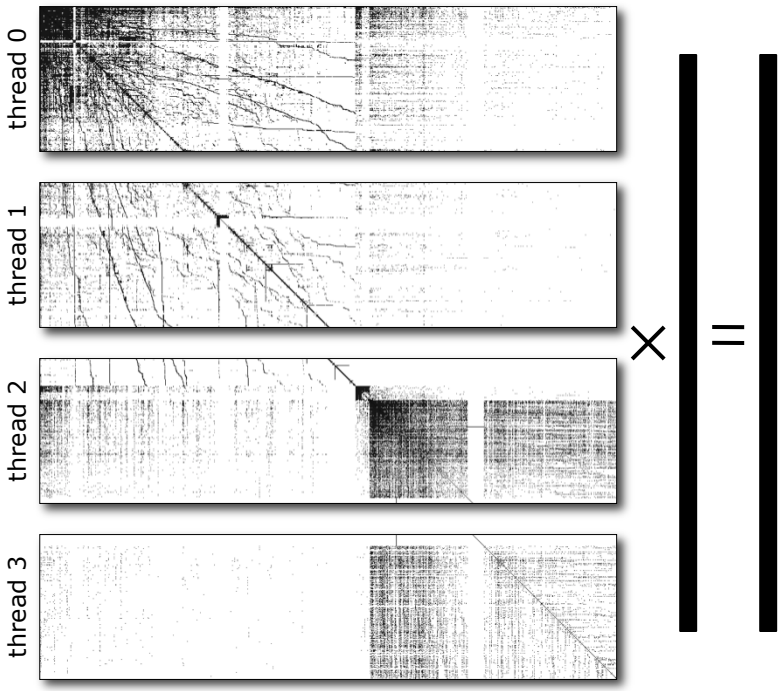
- Suite of 14 matrices
- All bigger than the caches of our SMPs
- We' ll also include a median performance number



Source: Sam Williams

SpMV Parallelization

- How do we parallelize a matrix-vector multiplication ?
- By rows blocks, load balance by number of nonzeros
- No inter-thread data dependencies, but random access to x

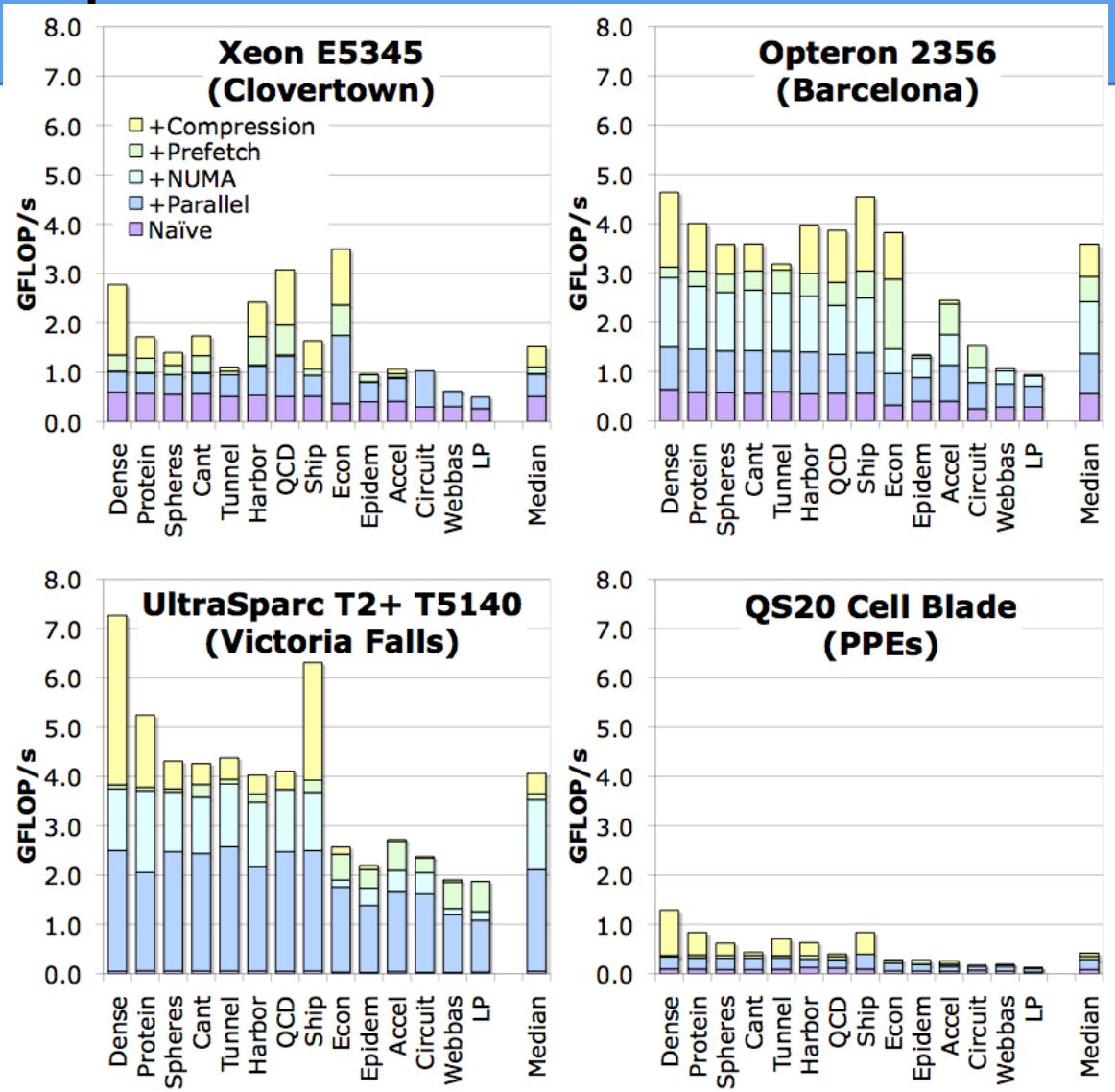


Source: Sam Williams

Summary of Multicore Optimizations

- NUMA - Non-Uniform Memory Access
  - pin submatrices to memories close to cores assigned to them
  - either explicit (malloc, affinity) or implicit (first touch)
- Prefetch – values, indices, and/or vectors
  - Pragma inserted in C code – special HW instructions
  - use exhaustive search on prefetch distance
- Matrix Compression – not just register blocking (BCSR)
  - 32 or 16-bit indices, Block Coordinate format for submatrices
- Cache-blocking
  - 2D partition of matrix, so needed parts of x,y fit in cache

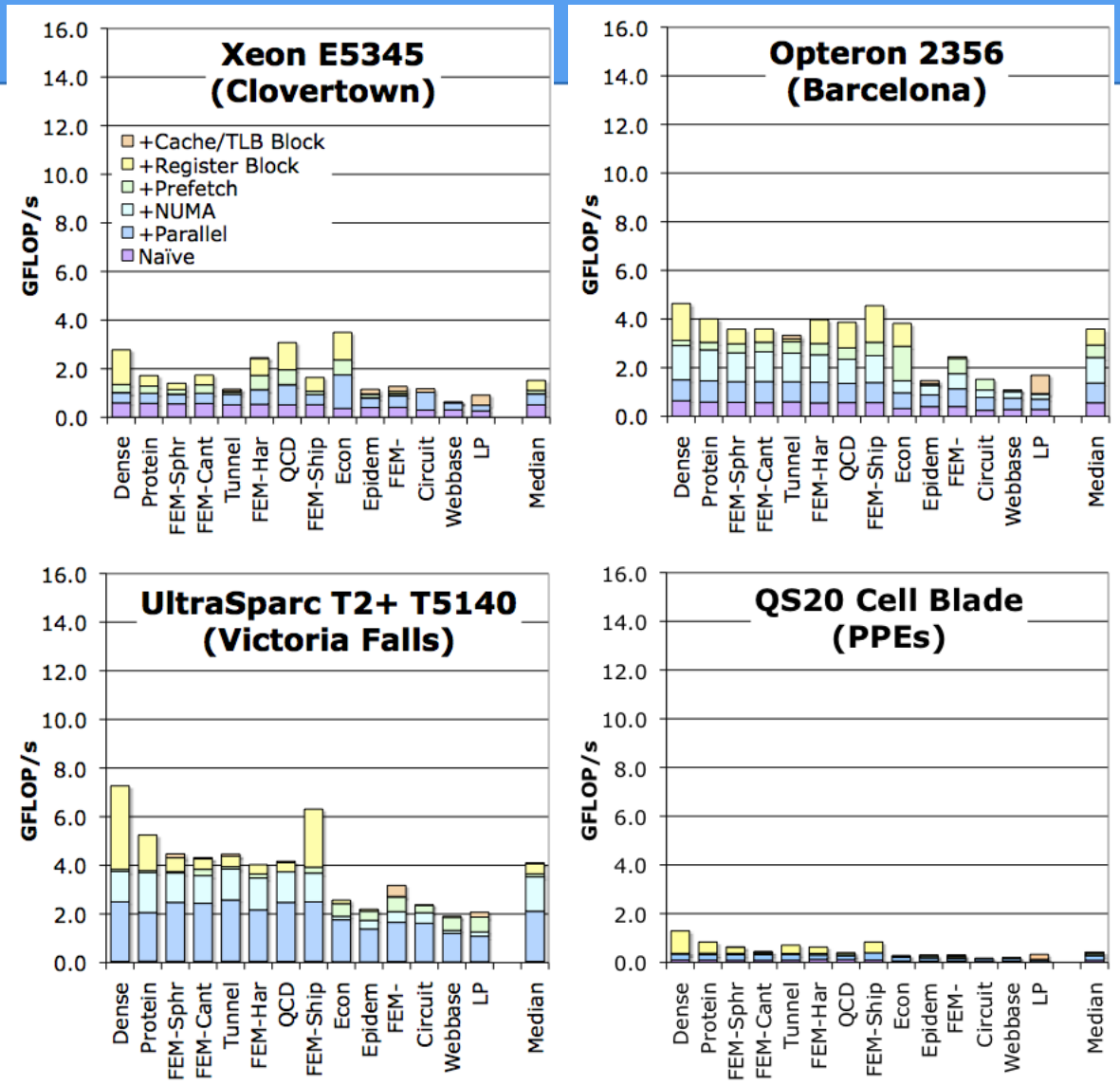
SpMV Performance



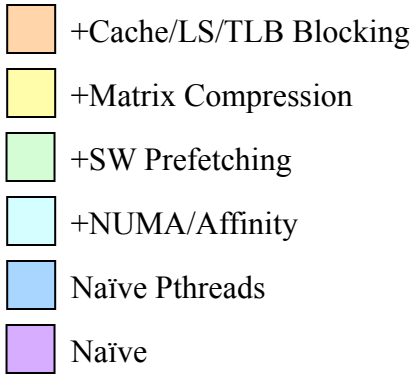
- After maximizing memory bandwidth, the only hope is to minimize memory traffic.
- Compression: exploit
  - register blocking
  - other formats
  - smaller indices
- Use a traffic minimization **heuristic** rather than search
- Benefit is clearly matrix-dependent.
- Register blocking enables efficient software prefetching (one per cache line)

Source: Sam Williams

Auto-tuned SpMV Performance
(cache and TLB blocking)

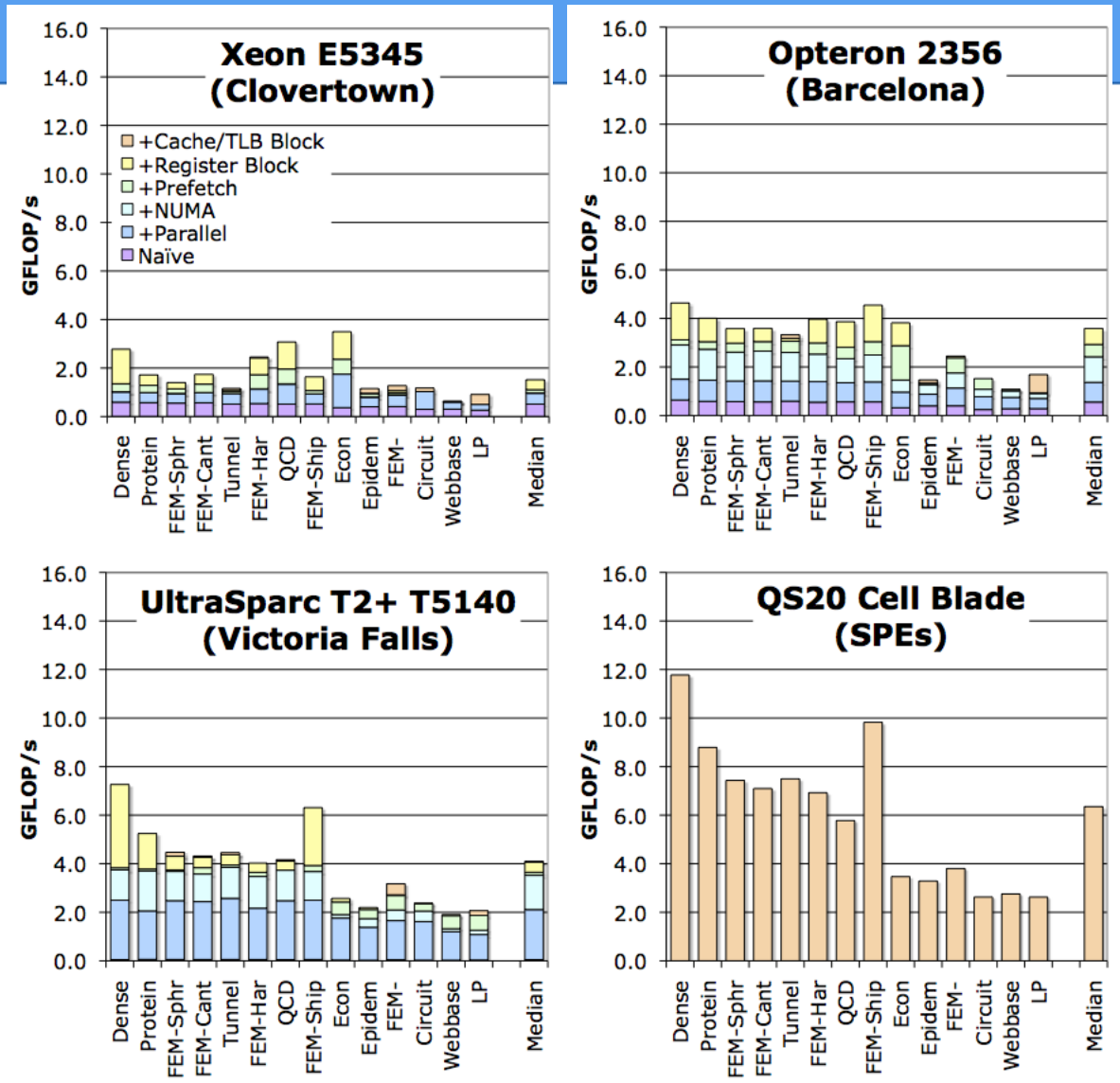


- Fully auto-tuned SpMV performance across the suite of matrices
- Why do some optimizations work better on some architectures?
- **matrices with naturally small working sets**
- **architectures with giant caches**



Source: Sam Williams

Auto-tuned SpMV Performance
(architecture specific optimizations)

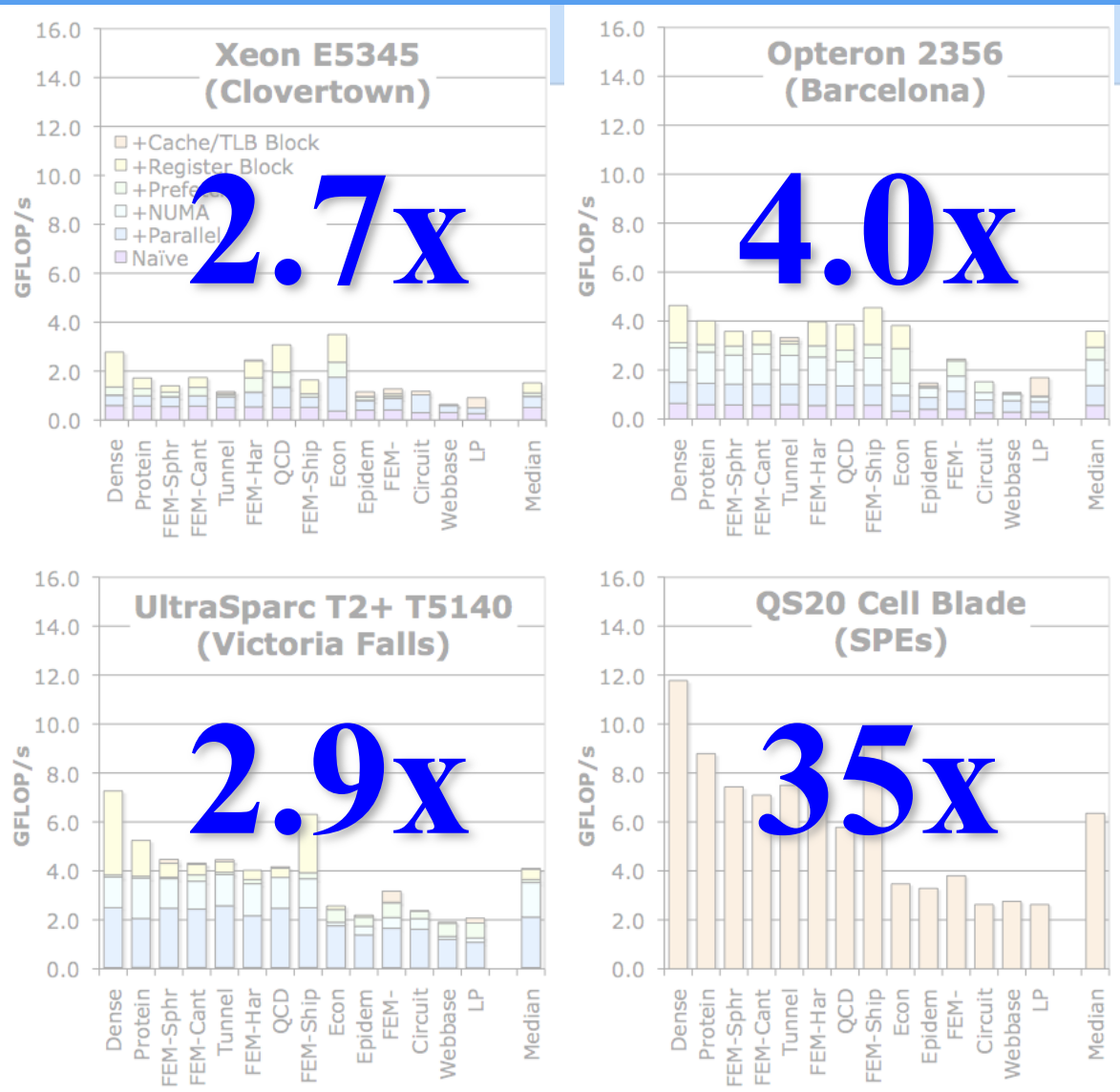


- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?

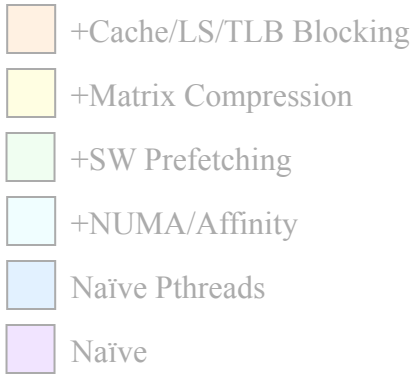
+Cache/LS/TLB Blocking
+Matrix Compression
+SW Prefetching
+NUMA/Affinity
Naïve Pthreads
Naïve

Source: Sam Williams

Auto-tuned SpMV Performance
(max speedup)



- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?



Source: Sam Williams

Plan

Krylov subspace methods

Tuning sparse matrix-vector product

Iterative solvers that reduce communication
- CA solvers based on s-step methods
- Enlarged Krylov methods

Preconditioners

Extra slides: one level preconditioners

Iterative solvers that reduce communication

Communication avoiding based on s-step methods

- Unroll k iterations, orthogonalize every k steps.
- A factor of $O(k)$ less messages and bandwidth in sequential.
- A factor of $O(k)$ less messages in parallel (same bandwidth).

Enlarged Krylov methods

- Decrease the number of iterations to decrease the number of global communications.
- Increase arithmetic intensity.

Other approaches available in the litterature, but not presented here.

CA solvers based on s-step methods: main idea

To avoid communication, unroll k-steps, ghost necessary data,

- generate a set of vectors W for the Krylov subspace $\mathcal{K}_k(A, r_0)$,
- (A)-orthogonalize the vectors using a communication avoiding orthogonalization algorithm (e.g. TSQR(W)).

References

- Van Rosendale '83, Walker '85, Chronopoulous and Gear '89, Erhel '93, Toledo '95, Bai, Hu, Reichel '91 (Newton basis), Joubert and Carey '92 (Chebyshev basis), etc.
- Recent references: G. Atenekeng, B. Philippe, E. Kamgnia (to enable multiplicative Schwarz preconditioner), J. Demmel, M. Hoemmen, M. Mohiyuddin, K. Yellick (to minimize communication, next slides), Carson, Demmel, Knight (CA and other Krylov solvers, preconditioners)

CA-GMRES

GMRES: find x in $span\{b, Ab, ..., A^k b\}$ minimizing $||Ax - b||_2$
Cost of k steps of standard GMRES vs new GMRES

Standard GMRES
for i=1 to k
w = A · v(i-1)
MGS(w, v(0),...,v(i-1))
update v(i), H
endfor
solve LSQ problem with H

Sequential: #words_moved =
O(k·nnz) from SpMV
+ O(k^2·n) from MGS
Parallel: #messages =
O(k) from SpMV
+ O(k^2 · log p) from MGS

Slide source: J. Demmel

CA-GMRES

GMRES: find x in $span\{b, Ab, ..., A^k b\}$ minimizing $||Ax - b||_2$
Cost of k steps of standard GMRES vs new GMRES

Standard GMRES
for i=1 to k
w = A · v(i-1)
MGS(w, v(0),...,v(i-1))
update v(i), H
endfor
solve LSQ problem with H

Communication-avoiding GMRES
W = [v, Av, A^2v, ... , A^kv]
[Q,R] = TSQR(W) ... "Tall Skinny QR"
Build H from R, solve LSQ problem

Sequential: #words_moved =
O(k·nnz) from SpMV
+ O(k^2·n) from MGS
Parallel: #messages =
O(k) from SpMV
+ O(k^2 · log p) from MGS

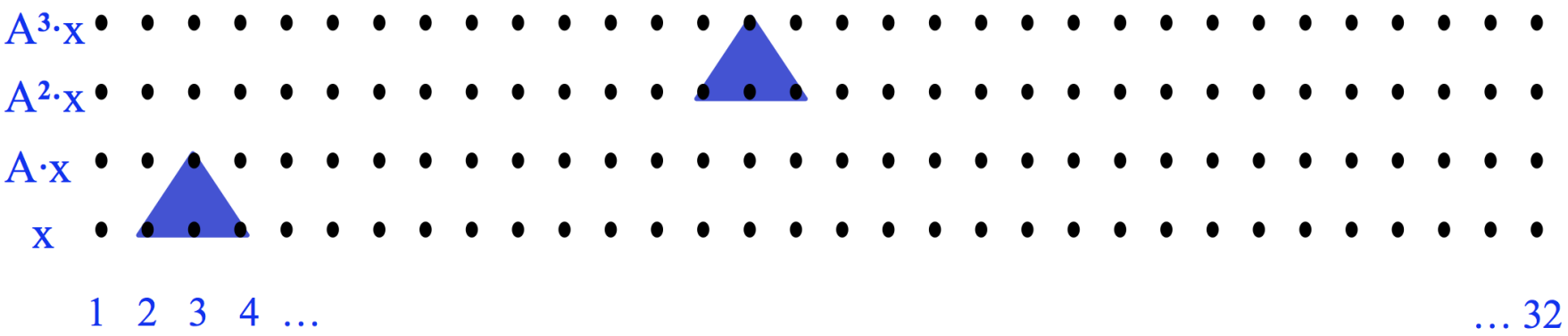
Sequential: #words_moved =
O(nnz) from SpMV
+ O(k·n) from TSQR
Parallel: #messages =
O(1) from computing W
+ O(log p) from TSQR

Slide source: J. Demmel

Matrix Powers Kernel

- Generate the set of vectors $\{Ax, A^2x, \ldots A^kx\}$ in parallel
- Ghost necessary data to avoid communication
- Example: A tridiagonal, $n = 32$, $s = 3$
- Shaded triangles represent data computed redundantly

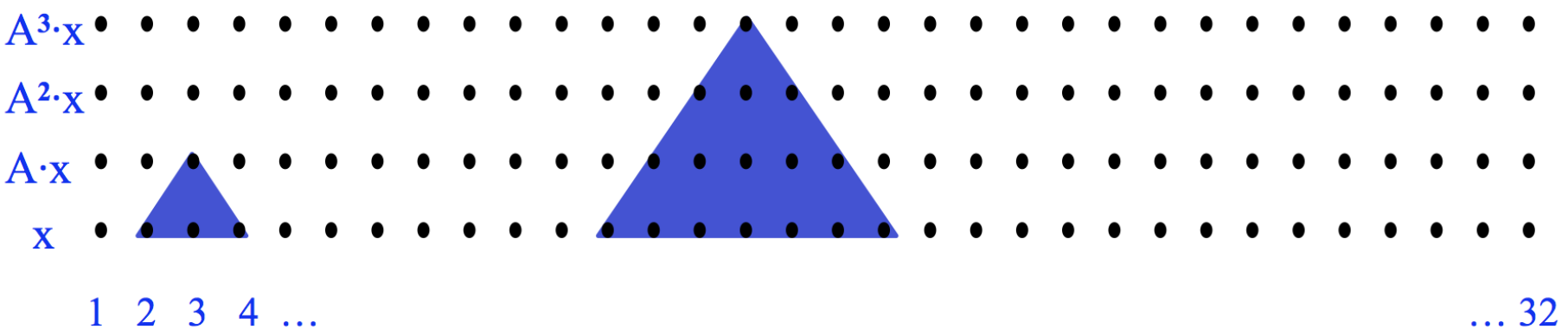
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



Matrix Powers Kernel

- Generate the set of vectors $\{Ax, A^2x, \ldots A^kx\}$ in parallel
- Ghost necessary data to avoid communication
- Example: A tridiagonal, $n = 32$, $s = 3$
- Shaded triangles represent data computed redundantly

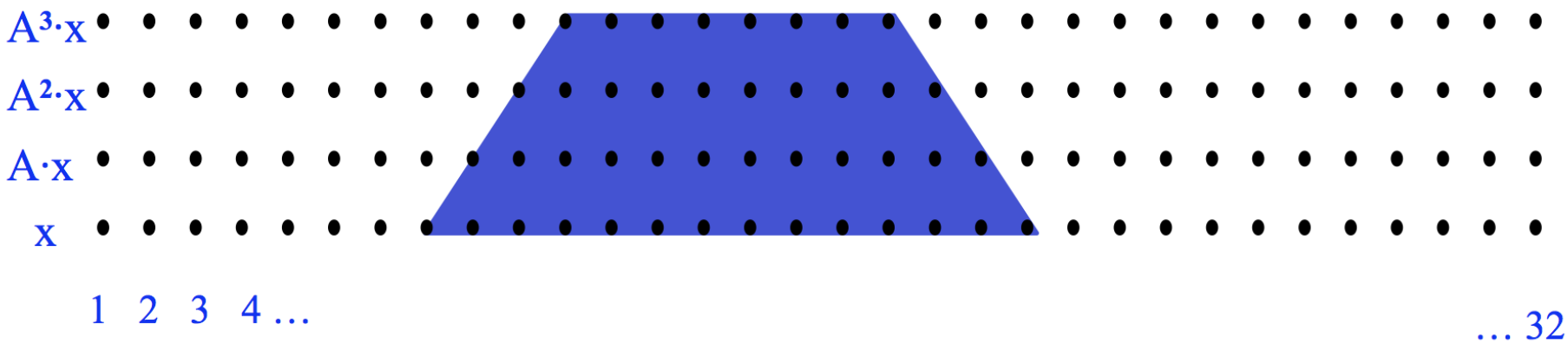
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



Matrix Powers Kernel

- Generate the set of vectors $\{Ax, A^2x, \ldots A^kx\}$ in parallel
- Ghost necessary data to avoid communication
- Example: A tridiagonal, $n = 32$, $s = 3$
- Shaded triangles represent data computed redundantly

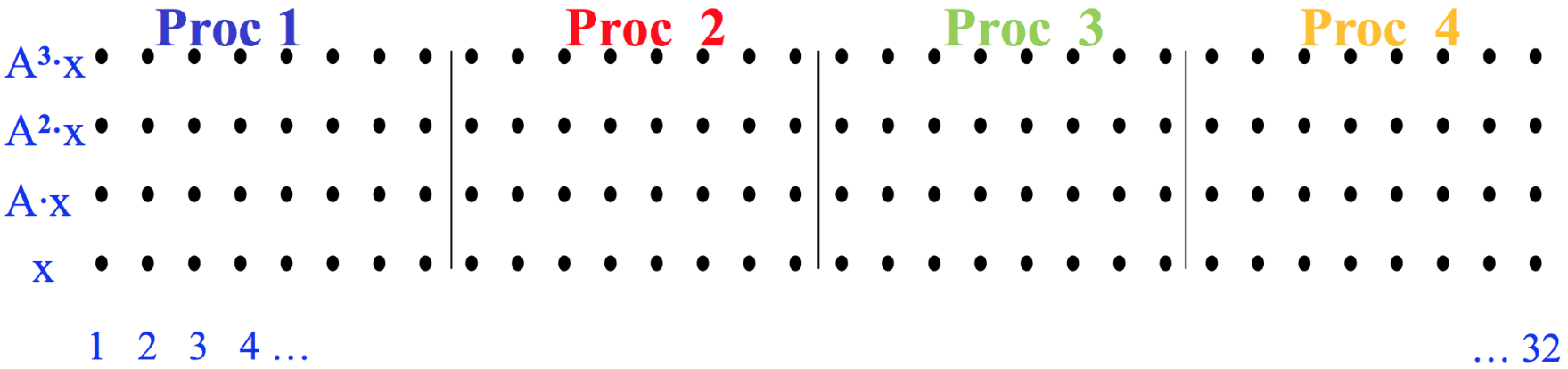
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



Matrix Powers Kernel

- Generate the set of vectors $\{Ax, A^2x, \ldots A^kx\}$ in parallel
- Ghost necessary data to avoid communication
- Example: A tridiagonal, $n = 32$, $s = 3$
- Shaded triangles represent data computed redundantly

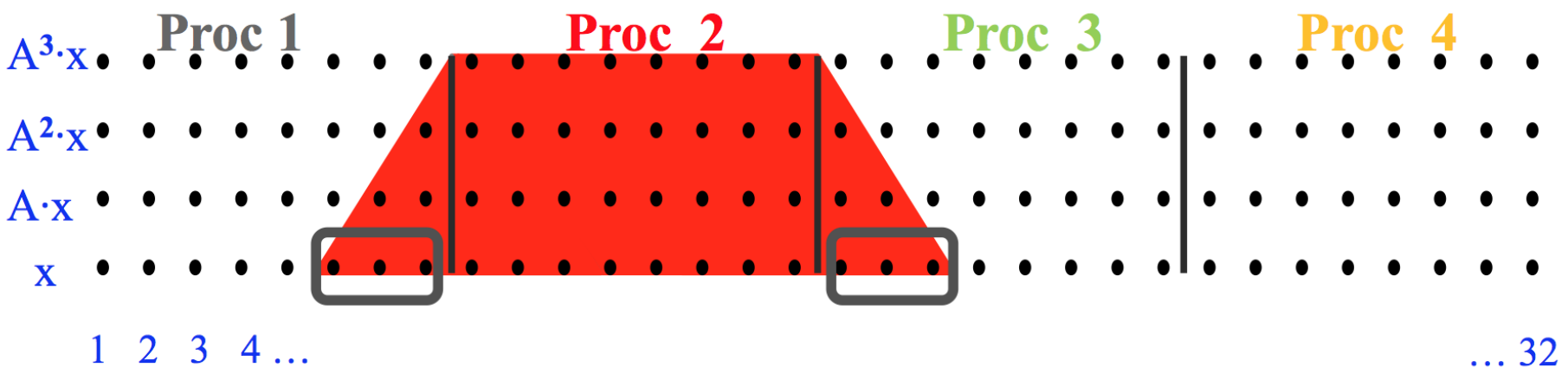
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



Matrix Powers Kernel

- Generate the set of vectors $\{Ax, A^2x, \ldots A^kx\}$ in parallel
- Ghost necessary data to avoid communication
- Example: A tridiagonal, $n = 32$, $s = 3$
- Shaded triangles represent data computed redundantly

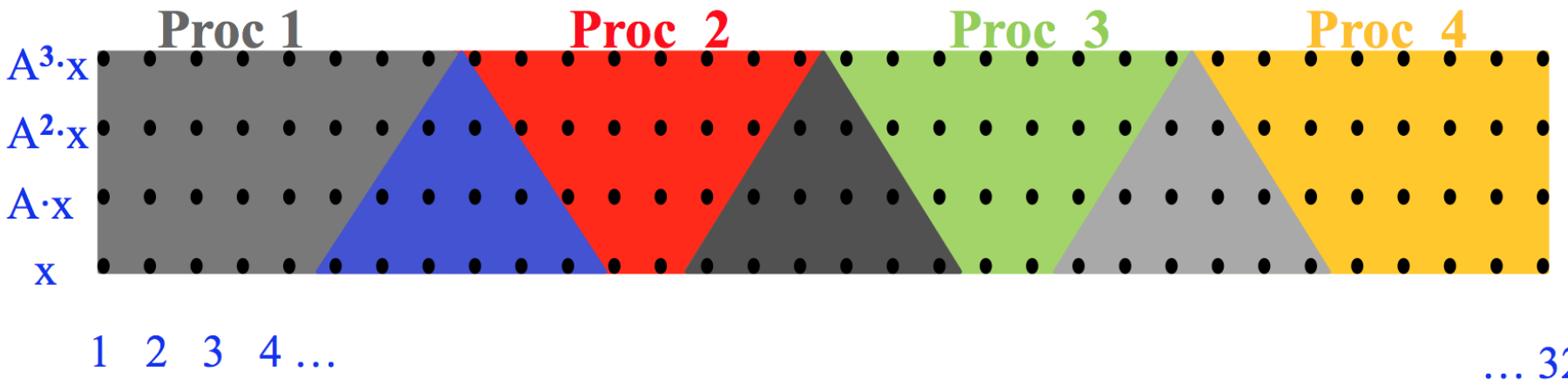
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



Matrix Powers Kernel

- Generate the set of vectors $\{Ax, A^2x, \ldots A^kx\}$ in parallel
- Ghost necessary data to avoid communication
- Example: A tridiagonal, $n = 32$, $s = 3$
- Shaded triangles represent data computed redundantly

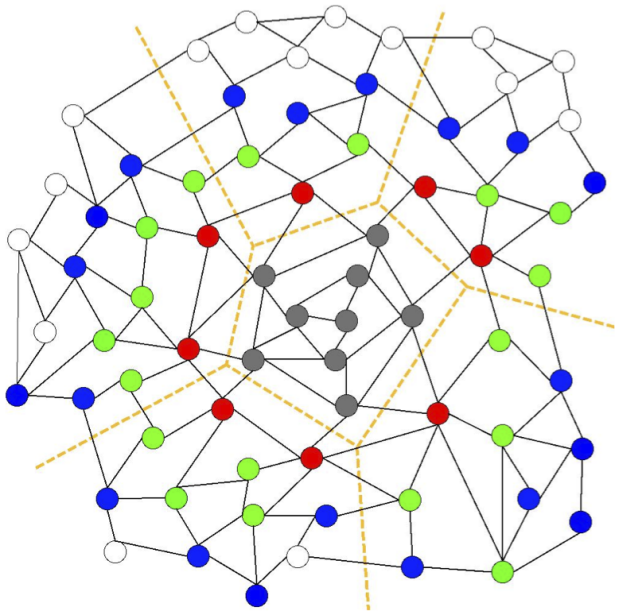
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



Matrix Powers Kernel (contd)

Ghosting works for structured or well-partitioned unstructured matrices, with modest surface-to-volume ratio.

- Parallel: block-row partitioning based on (hyper)graph partitioning,
- Sequential: top-to-bottom processing based on traveling salesman problem.



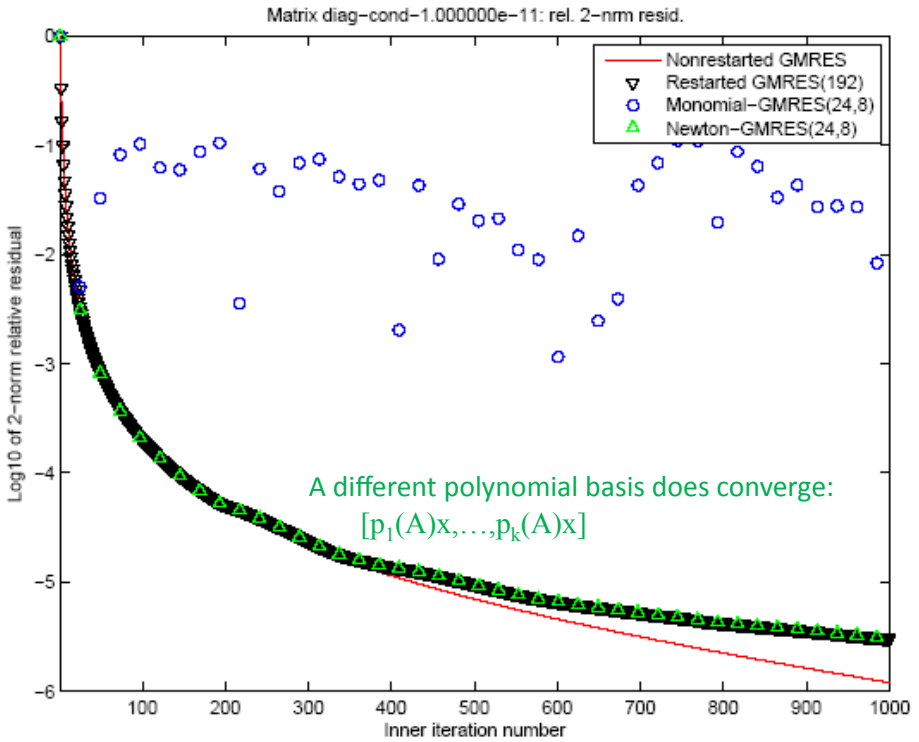
Challenges and research opportunities

Length of the basis k is limited by

- Size of ghost data
- Loss of precision

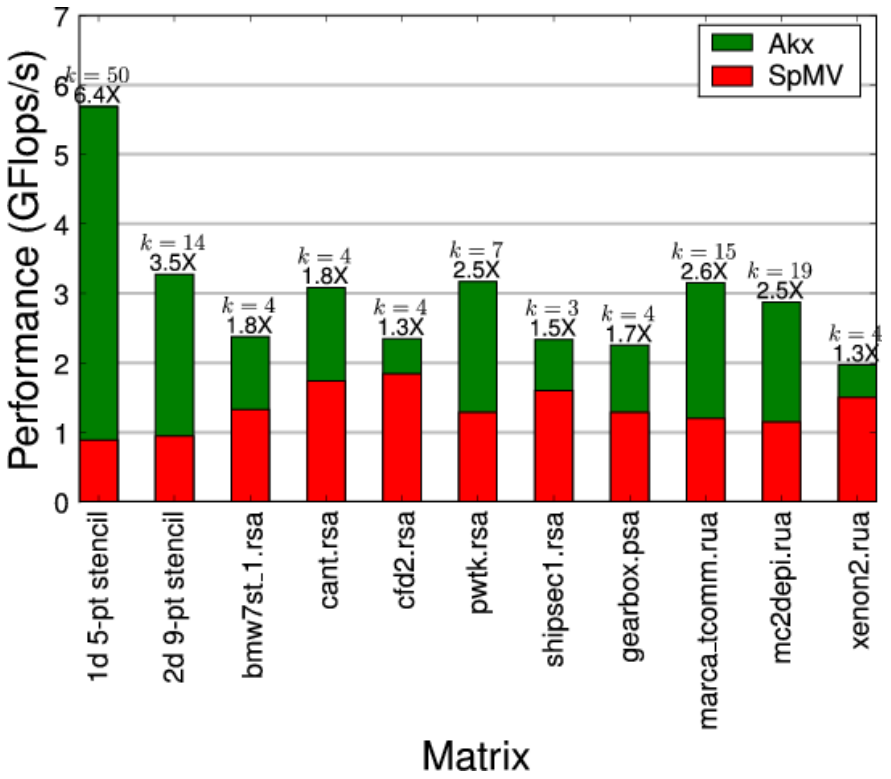
Preconditioners: lots of recent work

- Highly decoupled preconditioners: Block Jacobi
- Hierarchical, semiseparable matrices (M. Hoemmen, J. Demmel)
- CA-ILU0 (extra slides), deflation (Carson, Demmel, Knight)

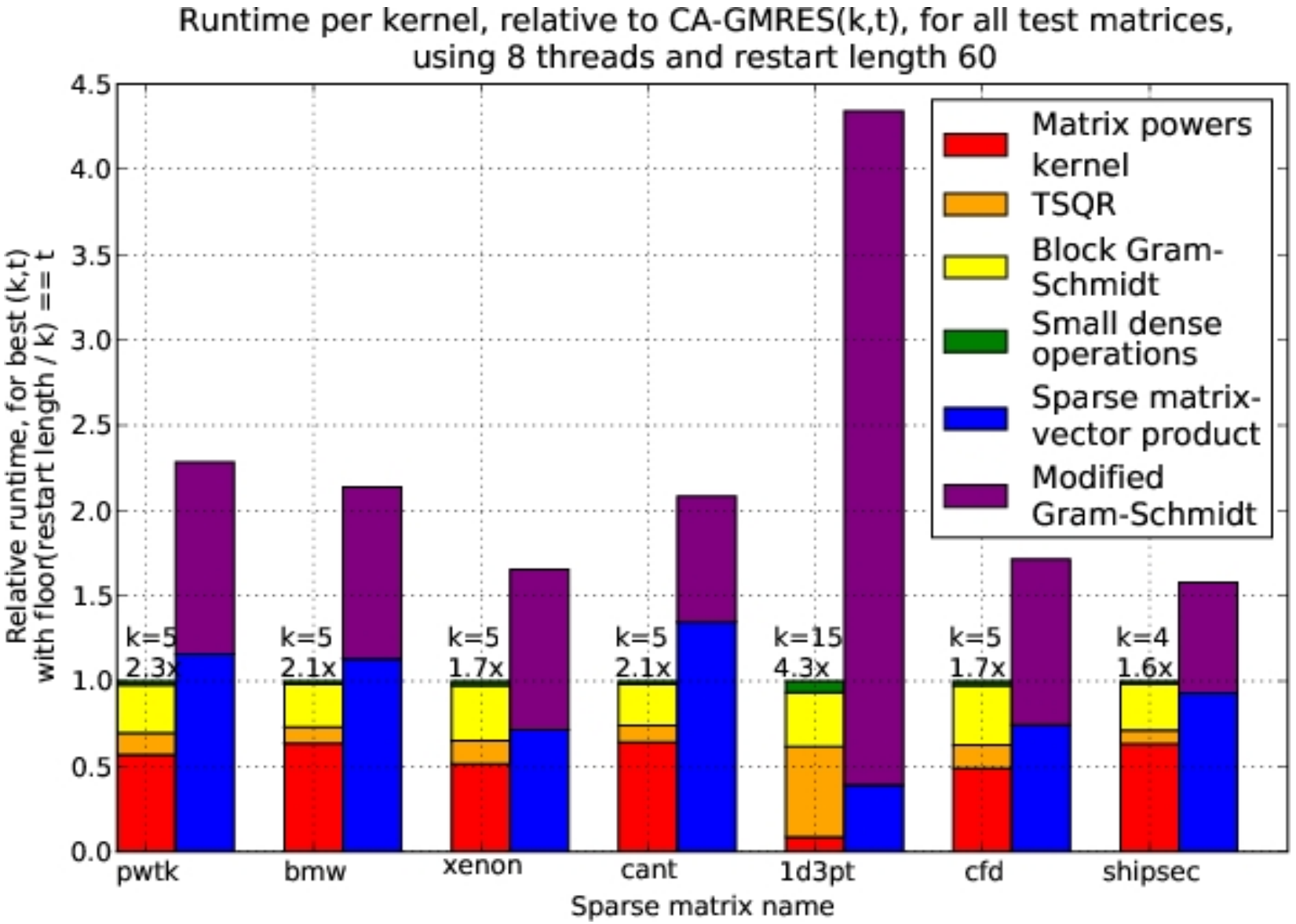


Performance

- Speedups on Intel Clovertown (8 cores), data from [Demmel et al., 2009]
- Used both optimizations:
  - sequential (moving data from DRAM to chip)
  - parallel (moving data between cores on chip)



Performance (contd)



Enlarged Krylov methods [Grigori et al., 2014]

- Partition the matrix into t domains
- At k-th iteration,
  - split the residual r_{k-1} into t vectors corresponding to the t domains,

$$r_{k-1} \rightarrow T(r_{k-1}) = \begin{bmatrix} * & 0 & & 0 \\ \vdots & \vdots & & \vdots \\ * & 0 & & 0 \\ 0 & * & & 0 \\ \vdots & \vdots & & \vdots \\ 0 & * & & 0 \\ & & \ddots & \\ 0 & 0 & & * \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & * \end{bmatrix}, T_s(r_{k-1}) = \{T(r_{k-1})(:,1), \ldots T(r_{k-1})(:,t)$$

  - generate t new basis vectors, obtain an enlarged Krylov subspace

$$\mathscr{K}_{t,k}(A, r_0) \quad = \quad span\{T_s(r_0), AT_s(r_0), A^2 T_s(r0), ..., A^{k-1} T_s(r_0)\}$$

  - search for the solution of the system $Ax = b$ in $\mathscr{K}_{t,k}(A, r_0)$

Properties of enlarged Krylov subspaces

- The Krylov subspace $\mathcal{K}_k(A, r_0)$ is a subset of the enlarged one

$$\mathcal{K}_k(A, r_0) \subset \mathscr{K}_{t,k}(A, r_0)$$

- For all $k < k_{max}$ the dimensions of $\mathscr{K}_{t,k}$ and $\mathscr{K}_{t,k+1}$ are stricltly increasing by some number i_k and i_{k+1} respectively, where

$$t \geq i_k \geq i_{k+1} \geq 1.$$

- The enlarged subspaces are increasing subspaces, yet bounded.

$$\mathscr{K}_{t,1}(A, r_0) \subsetneq \ldots \subsetneq \mathscr{K}_{t,k_{max}-1}(A, r_0) \subsetneq \mathscr{K}_{t,k_{max}}(A, r_0) = \mathscr{K}_{t,k_{max}+q}(A, r_0), \forall q > 0$$

Properties of enlarged Krylov subspaces: stagnation

- Let $\mathcal{K}_{p_{max}} = \mathcal{K}_{p_{max}+q}$ and $\mathscr{K}_{t,k_{max}} = \mathscr{K}_{t,k_{max}+q}$ for $q > 0$. Then

$$k_{max} \leq p_{max}.$$

- The solution of the system $Ax = b$ belongs to the subspace $x_0 + \mathscr{K}_{t,k_{max}}$.

Enlarged Krylov subspace methods based on CG

Defined by the subspace $\mathscr{K}_{t,k}$ and the following two conditions:

1. Subspace condition: $x_k \in x_0 + \mathscr{K}_{t,k}$
2. Orthogonality condition: $r_k \perp \mathscr{K}_{t,k}$

- At each iteration, the new approximate solution x_k is found by minimizing $\phi(x) = \frac{1}{2}(x)^t Ax - b^t x$ over $x_0 + \mathscr{K}_{t,k}$:

$$\phi(x_k) = min\{\phi(x), \forall x \in x_0 + \mathscr{K}_{t,k}(A, r_0)\}$$

Convergence analysis

Given

- A is an SPD matrix, x^* is the solution of $Ax = b$
- $||\overline{e}_k||_A = ||x^* - \overline{x}_k||_A$ is the k^{th} error of CG
- $||e_k||_A = ||x^* - x_k||_A$ is the k^{th} error of enlarged methods
- CG converges in $\overline{K}$ iterations

Result

Enlarged Krylov methods converge in K iterations, where $K \leq \overline{K} \leq n$.

$$||e_k||_A = ||x^* - x_k||_A \leq ||\overline{e}_k||_A$$

LRE-CG: Long Recurrence Enlarged CG

- Use the entire basis to approximate the new solution
- $Q_k = [W_1 W_2 \ldots W_k]$ is an $n \times tk$ matrix containing the basis vectors of $\mathscr{K}_{t,k}$
- At each k^{th} iteration, approximate the solution as

$$x_k = x_{k-1} + Q_k \alpha_k$$

such that

$$\phi(x_k) = min\{\phi(x), \forall x \in x_0 + \mathscr{K}_{t,k}\}$$

- Either x_k is the solution, or t new basis vectors and the new approximation $x_{k+1} = x_k + Q_{k+1}\alpha_{k+1}$ are computed.

SRE-CG: Short recurrence enlarged CG

- By A-orthonormalizing the basis vectors $Q_k = [W_1, W_2, \ldots W_k]$, we obtain a short recurrence enlarged CG.
- Given that $Q_{k-1}^t r_{k-1} = 0$, we obtain the recurrence relations:

$$\begin{aligned} \alpha_k &= W_k^t r_{k-1}, \\ x_k &= x_{k-1} + W_k \alpha_k, \\ r_k &= r_{k-1} - A W_k \alpha_k, \end{aligned}$$

- W_k needs to be A-orthormalized only against W_{k-1} and W_{k-2}.

SRE-CG Algorithm

Algorithm 2 The SRE-CG algorithm

Input: A, b, x_0, ϵ, k_{max}
Output: x_k, the approximate solution of the system $Ax = b$

$r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2^2$, $k = 1$
while ($\sqrt{\rho_{k-1}} > \epsilon||b||_2$ and $k < k_{max}$) **do**
    if k==1 **then**
        Let $W_1 = T(r_0)$, A-orthonormalise its vectors
    else
        Let $W_k = AW_{k-1}$
        A-orthonormalise W_k against W_{k-1} and W_{k-2} if $k > 2$
        A-orthonormalise the vectors of W_k
    end if
    $\alpha_k = (W_k^t r_{k-1})$
    $x_k = x_{k-1} + W_k \alpha_k$
    $r_k = r_{k-1} - AW_k \alpha_k$
    $\rho_k = ||r_k||_2^2$
    k = k+1
end while

SRE-CG: cost on t processors

Cost of $\bar{k}$ iterations of CG is:

$$\begin{array}{rcl} \text{Total Flops} & \approx & 2nnz \cdot \bar{k}/t + 4n\bar{k}/t \\ \text{\# words} & \approx & O(\bar{k}) \text{ (from SpMV)} \\ \text{\# messages} & \approx & 2 \text{ k log(t)} + \text{O(k) (from SpMV)} \end{array}$$

Cost of k iterations of SRE-CG is:

$$\begin{array}{rcl} \text{Total Flops} & \approx & 2nnz \cdot k + O(ntk) \\ \text{\# words} & \approx & kt^2 log(t) + O(k) \text{ (from SpMV)} \\ \text{\# messages} & \approx & klog(t) + \text{O(k) (from SpMV)} \end{array}$$

Ideally, SRE-CG converges t times faster ($k = \bar{k}/t$)
$\Rightarrow$ SRE-CG has a factor of $\bar{k}/k$ less global communication.

Convergence of different CG versions

	CG		SRE-CG	
Pa	Iter	Err	Iter	Err
SKY3D				
8	902	1E-5	211	1E-5
16	902	1E-5	119	9E-6
32	902	1E-5	43	4E-6
ANI3D				
2	4187	4e-5	875	7e-5
4	4146	4e-5	673	8e-5
8	4146	4e-5	449	1e-4
16	4146	4e-5	253	2e-4
32	4146	4e-5	148	2e-4
64	4146	4e-5	92	1e-4
ELAST3D				
2	1098	1e-7	652	1e-7
4	1098	1e-7	445	1e-7
8	1098	1e-7	321	8e-8
16	1098	1e-7	238	4e-8
32	1098	1e-7	168	5e-8
64	1098	1e-7	116	1e-8

Plan

Krylov subspace methods

Tuning sparse matrix-vector product

Iterative solvers that reduce communication

Preconditioners

- One level preconditioners: CA-ILU0
- Two level preconditioners

Extra slides: one level preconditioners

Preconditioned Krylov subspace methods

- Solve by using iterative methods

$$Ax = b.$$

- Convergence depends on $\kappa(A)$ and the eigenvalue distribution (for SPD matrices).
- To accelerate convergence, solve

$$M^{-1}Ax = M^{-1}b,$$

  where
  - M approximates well the inverse of A and/or
  - improves $\kappa(A)$, the condition number of A.
- Ideally, we would like to bound $\kappa(A)$, independently of the size of the matrix A.

One level preconditioners (two examples)

Incomplete LU factorization

- Computes $A = LU + E$
- Preconditioner $M = LU$
- ILU0 does not introduce any fill in the factors

Block Jacobi preconditioner

Given

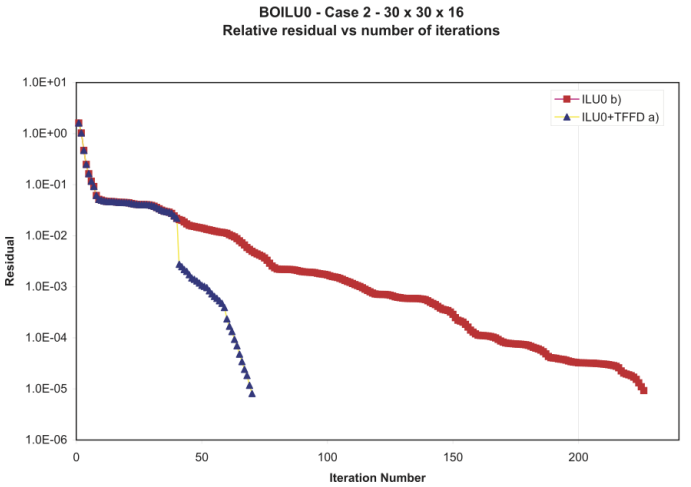
$$A = \begin{pmatrix} A_{11} & \dots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \dots & A_{PP} \end{pmatrix}$$

block Jacobi preconditioner is:

$$M = \begin{pmatrix} A_{11} & & \\ & \ddots & \\ & & A_{PP} \end{pmatrix} = \begin{pmatrix} L_{11}\,U_{11} & & \\ & \ddots & \\ & & L_{PP}\,U_{PP} \end{pmatrix} = LU$$

The need for two level preconditioners

- When solving complex linear systems arising, e.g. from large discretized systems of PDEs with strongly heterogeneous coefficients (high contrast, multiscale).



- Flow in porous media
- Elasticity problems
- CMB data analysys (no PDE)

- Most of the existing preconditioners lack robustness
  - wrt jumps in coefficients / partitioning into irregular subdomains, e.g. one level DDM methods (block Jacobi, RAS), incomplete LU
  - A few small eigenvalues hinder the convergence of iterative methods

Using deflation to deal with low frequency modes

In the unified framework of [Tang et al., 2009], let :

$$P := I - AZE^{-1}Z^T, \quad E := Z^T AZ$$

where

- Z is the deflation subspace matrix of full rank
- E is the coarse grid correction, a small dense invertible matrix
- P is the deflation matrix, $PAZ = 0$

Usage in different classes of preconditioners

- DDM - Z and Z^T are the restriction and prolongation operators based on subdomains, E is a coarse grid, P is a subspace correction
- Deflation - Z contains the vectors to be deflated
- Multigrid - interpretation possible

Using deflation to deal with low frequency modes

In the unified framework of [Tang et al., 2009], let :

$$P := I - AZE^{-1}Z^T, \quad E := Z^T AZ$$

where

- Z is the deflation subspace matrix of full rank
- E is the coarse grid correction, a small dense invertible matrix
- P is the deflation matrix, $PAZ = 0$

Usage in different classes of preconditioners

- DDM - Z and Z^T are the restriction and prolongation operators based on subdomains, E is a coarse grid, P is a subspace correction
- Deflation - Z contains the vectors to be deflated
- Multigrid - interpretation possible

Using deflation to deal with low frequency modes

In the unified framework of [Tang et al., 2009], let :

$$P := I - AZE^{-1}Z^T, \quad E := Z^T AZ$$

where

- Z is the deflation subspace matrix of full rank
- E is the coarse grid correction, a small dense invertible matrix
- P is the deflation matrix, $PAZ = 0$

Example of preconditioner

$$P_{2lvl}^{-1} = M^{-1}P + ZE^{-1}Z^T,$$

where M is the first level preconditioner (eg based on block Jacobi).

- $P_{2lvl}^{-1}AZ = Z$
- The small eigenvalues are shifted to 1.
- P_{2lvl} is not SPD, even when A is, better choices available, but more expensive.

Using deflation to deal with low frequency modes

In the unified framework of [Tang et al., 2009], let :

$$P := I - AZE^{-1}Z^T, \quad E := Z^T AZ$$

where

- Z is the deflation subspace matrix of full rank
- E is the coarse grid correction, a small dense invertible matrix
- P is the deflation matrix, $PAZ = 0$

Example of preconditioner

$$P_{2lvl}^{-1} = M^{-1}P + ZE^{-1}Z^T,$$

where M is the first level preconditioner (eg based on block Jacobi).

- $P_{2lvl}^{-1}AZ = Z$
- The small eigenvalues are shifted to 1.
- P_{2lvl} is not SPD, even when A is, better choices available, but more expensive.

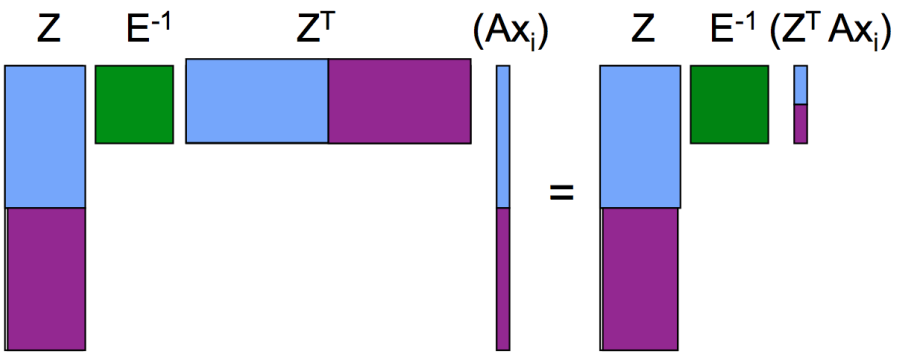
Two level preconditioners (contd)

Computing the preconditioner requires

- Deflation subspace Z, which can be formed by
  - Eigenvectors corresponding to smallest eigenvalues - from previous linear systems solved with different right hand sides, etc.
  - Using knowledge from the physics, partition of the unity, etc.
- Computing AZ and $E = Z^T AZ$.

Applying the preconditioner at each iteration requires

- Computing $y = ZE^{-1}Z^T(Ax_i) = ZE^{-1}Z^T v$
  $\Rightarrow$ involves collective communication when computing $Z^T v$, and solving a linear system with E.



Two level preconditioners (contd)

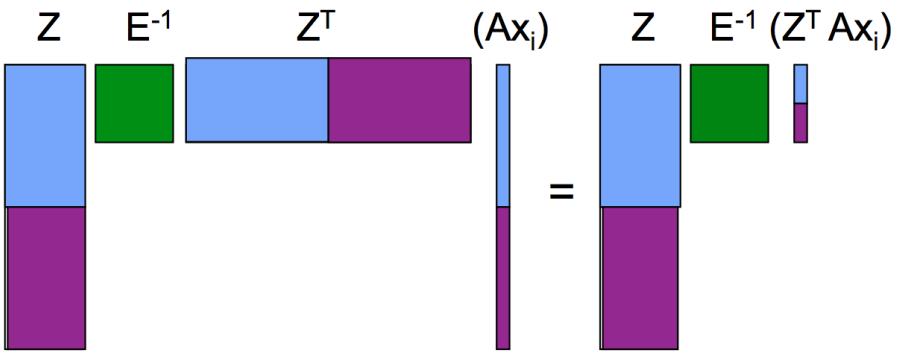
Computing the preconditioner requires

- Deflation subspace Z, which can be formed by
  - Eigenvectors corresponding to smallest eigenvalues - from previous linear systems solved with different right hand sides, etc.
  - Using knowledge from the physics, partition of the unity, etc.
- Computing AZ and $E = Z^T AZ$.

Applying the preconditioner at each iteration requires

- Computing $y = ZE^{-1}Z^T(Ax_i) = ZE^{-1}Z^T v$

  $\Rightarrow$ involves collective communication when computing $Z^T v$, and solving a linear system with E.



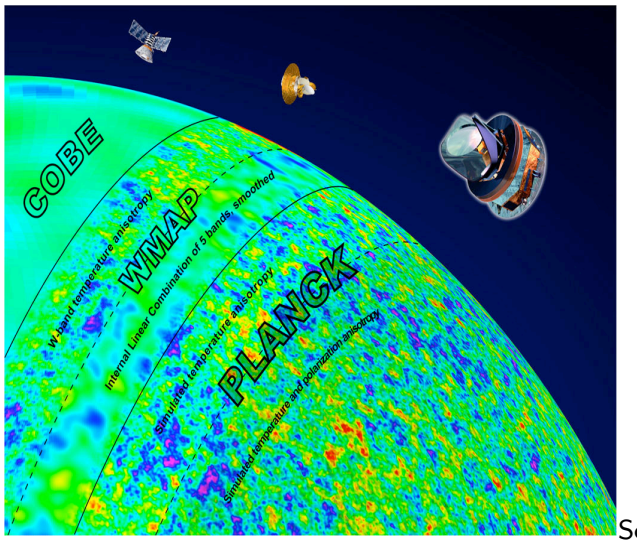
Example of deflation used in CMB data analysis

CMB data analysis

- Study light left over after the ever mysterious Big Bang,
- Produce and analyze multi-frequency 2D images of the universe when it was 5% of its current age.

- COBE (1989) collected 10 gigabytes of data, required 1 Teraflop per image analysis.
- PLANCK (2010) produced 1 terabyte of data, requires 100 Petaflops per image analysis.
- Future experiment (2020) estimated to collect .5 petabytes, require 100 Exaflops per image analysis.

Source: J. Borrill, LBNL, R. Stompor, Paris 7



Source: http://www.epm.ornl.gov/chammp/chammp.html

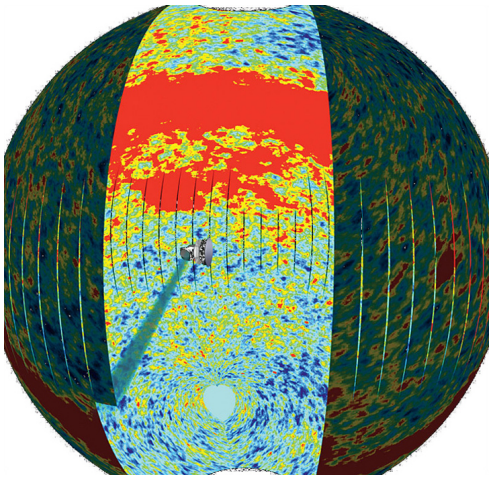
Map-making problem in an (algebraic) nutshell

- Find the best map x from observations d, scanning strategy A, and noise n_t

$$d = Ax + n_t$$

- Assuming the noise properties are Gaussian and piece-wise stationary, the covariance matrix is $N =< n_t n_t^T >$, and N^{-1} is a block diagonal symmetric Toeplitz matrix.
- The solution of the generalized least squares problem is found by solving

$$A^T N^{-1} A x = A^T N^{-1} d$$



Scanning strategy in our experiments:

- 2048 densely crossing circles
- Each circle is scanned 32 times, leading to 10^6 samples
- Piece-wise stationary noise, one Toeplitz block for each circle

Traditional approach used in the CMB community

- Solve the linear system using preconditioned CG:

$$\begin{aligned} M_{diag} S x &= M_{diag} b, \text{ where} \\ S &:= A^T N^{-1} A,\ b := A^T N^{-1} d,\ M_{diag} := (A^T diag(N^{-1}) A)^{-1} \end{aligned}$$

- The diagonal preconditioner M_{diag} does not scale numerically.

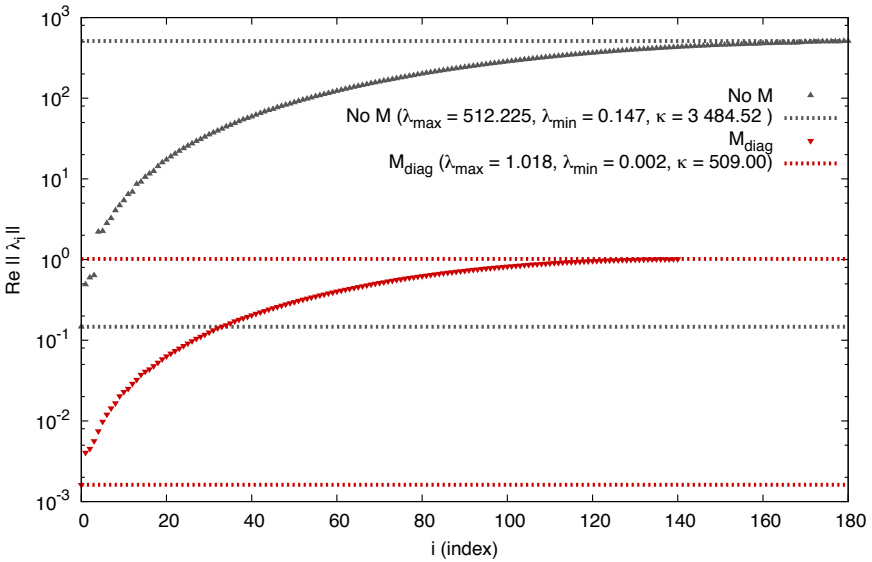


Figure : Eigenvalue distribution of S and $M_{diag}^{-1} S$ (NoM and M_{diag} resp. in the plot).

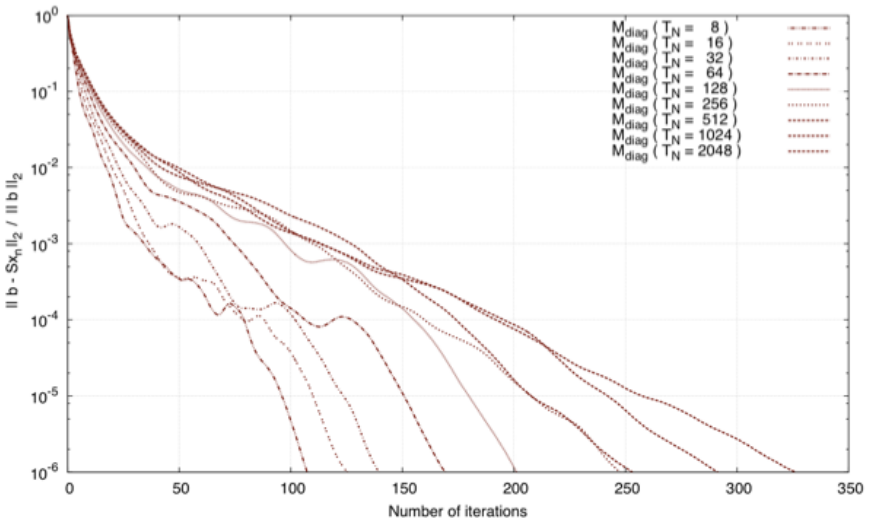


Figure : Convergence of preconditioned CG when increasing the size of the problem, e.g. number of circles T_N.

Two level preconditioner for the map-making problem

- Combine diagonal preconditioner with deflation

$$M_{2lvl} = M_{diag}(I - S(ZE^{-1}Z^T)) + ZE^{-1}Z^T,$$
$$\text{where } M_{diag} = (A^T diag(N^{-1})A)^{-1}, \quad E = Z^T SZ$$

- The efficiency of the preconditioner depends on the choice of Z see for more details [Grigori et al., 2012, Szydlarski et al., 2014].

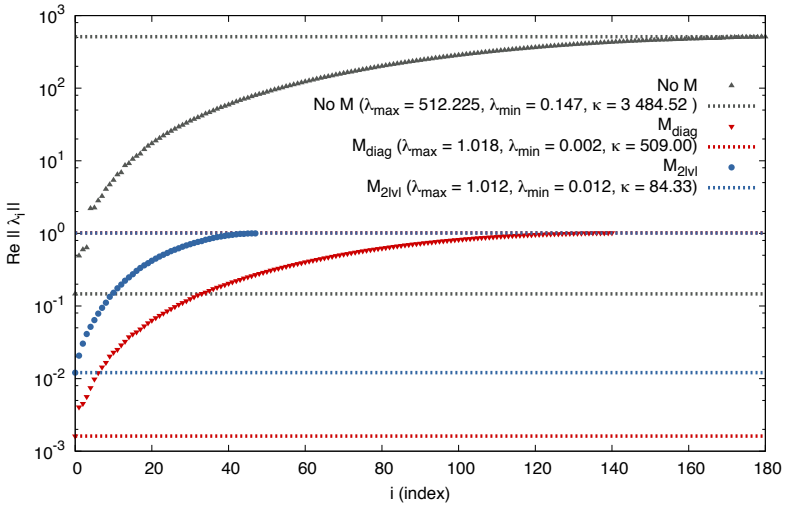


Figure : Eigenvalue distribution of S, $M_{diag}^{-1}S$, $M_{2lvl}^{-1}S$ (NoM, M_{diag}, M_{2lvl} resp. in the plot).

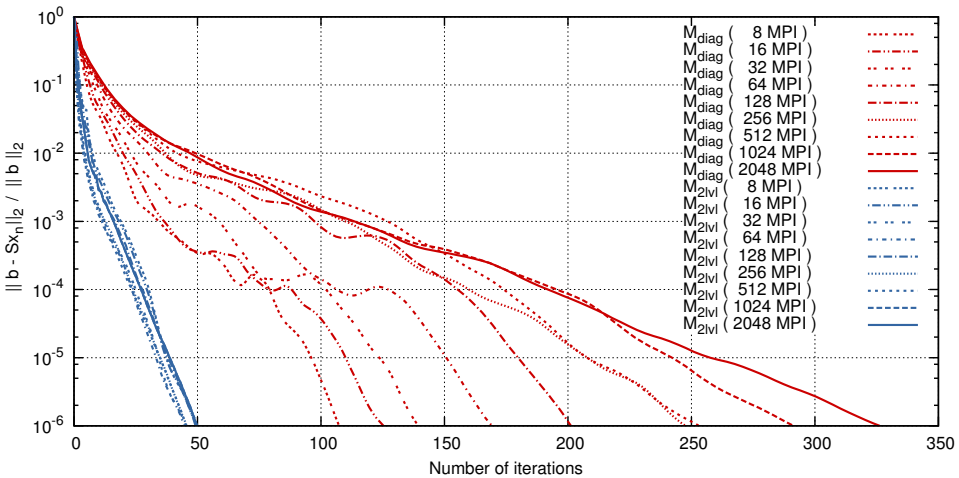
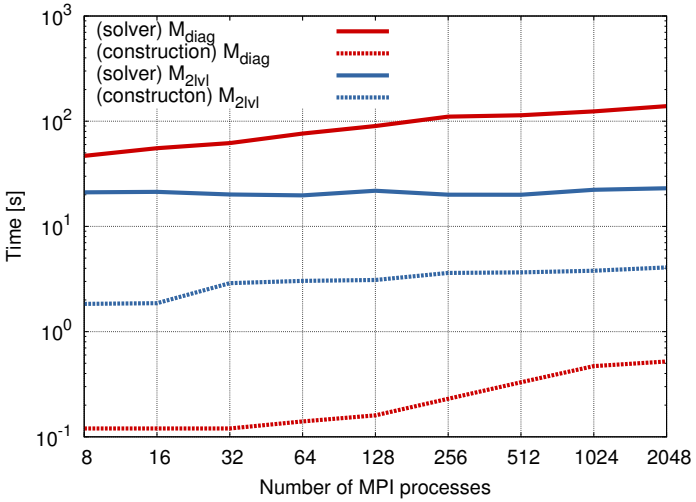
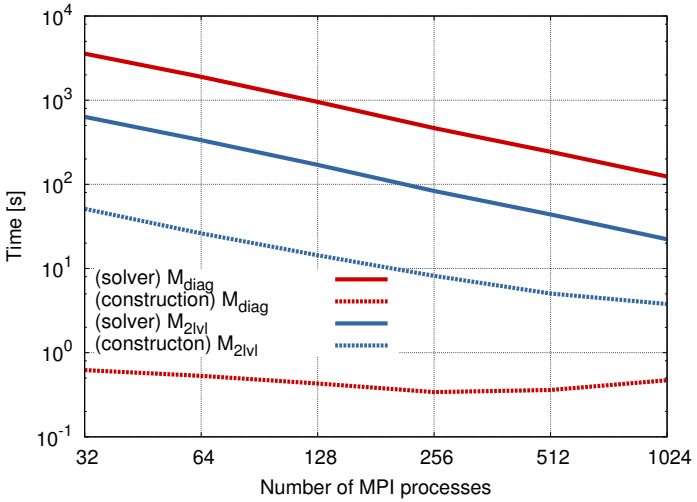


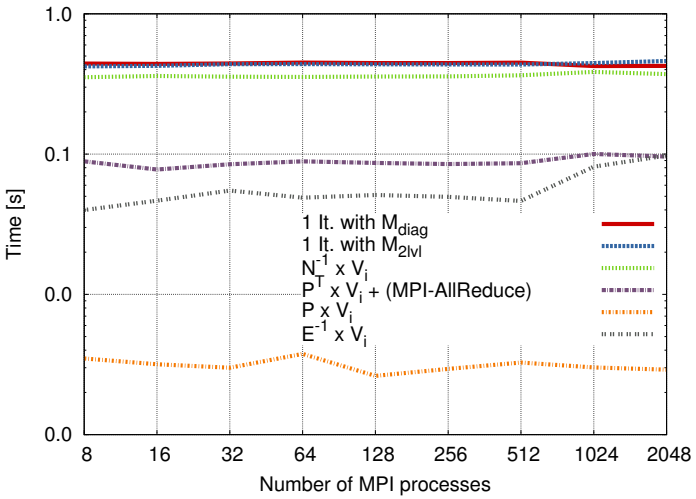
Figure : Convergence of preconditioned CG when increasing the size of the problem, number of circles = no of MPI processes.

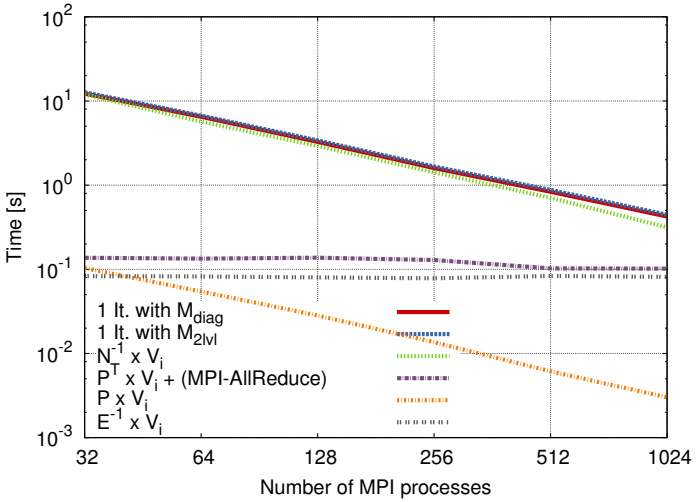
Timings for weak (left) and strong (right) scaling

- 1 or more (for strong scaling) circles per 1 MPI process.
- 1 MPI process mapped on 6 cores of NERSC's Hopper Cray XE6.









Plan

Krylov subspace methods

Tuning sparse matrix-vector product

Iterative solvers that reduce communication

Preconditioners

Extra slides: one level preconditioners

- One level preconditioners: examples

One level preconditioners (two examples)

Incomplete LU factorization

- Computes $A = LU + E$
- Preconditioner $M = LU$
- ILU0 does not introduce any fill in the factors

Block Jacobi preconditioner

Given

$$A = \begin{pmatrix} A_{11} & \dots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \dots & A_{PP} \end{pmatrix}$$

block Jacobi preconditioner is:

$$M = \begin{pmatrix} A_{11} & & \\ & \ddots & \\ & & A_{PP} \end{pmatrix} = \begin{pmatrix} L_{11}\,U_{11} & & \\ & \ddots & \\ & & L_{PP}\,U_{PP} \end{pmatrix} = LU$$

Left-preconditioned system

- A preconditioned matrix powers kernel computes the set basis vectors

$$\{M^{-1}Ay_0, (M^{-1}A)^2y_0, ..., (M^{-1}A)^{s-1}y_0, (M^{-1}A)^s y_0\}$$

  where y_0 is a starting vector and $s \geq 1$.

- The i-th iteration of a Krylov subspace solver preconditioned with $M = LU$ computes $y_i = (LU)^{-1}Ay_{i-1}$ as:

  1. Compute $f = Ay_{i-1}$
  2. Solve $LUy_i = f$ i.e.

     2.1 Solve $Lz = f$ by forward substitution

     2.2 Solve $Uy_i = z$ by backward substitution

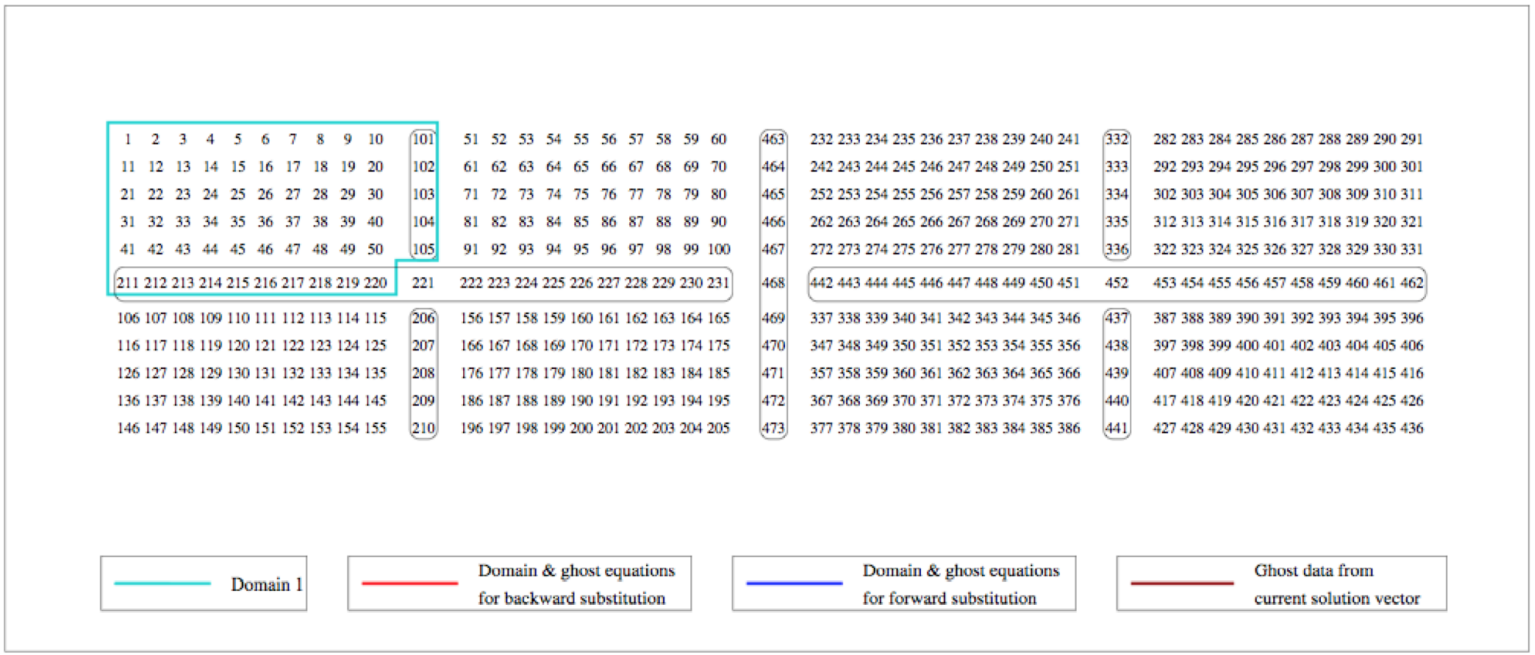
ILU0 with nested dissection

Can we compute s iterations with no communication ?

Compute $y_i = (LU)^{-1} A y_{i-1}$ using 3 steps:

1. Compute $f = A y_{i-1}$
2. Solve $Lz = f$ by forward substitution
3. Solve $U y_i = z$ by backward substitution

$$\begin{pmatrix} A_{11} & & A_{13} & & & & A_{17} \\ & A_{22} & A_{23} & & & & A_{27} \\ A_{31} & A_{32} & A_{33} & & & & A_{37} \\ & & & A_{44} & & A_{46} & A_{47} \\ & & & & A_{55} & A_{56} & A_{57} \\ & & & A_{64} & A_{65} & A_{66} & A_{67} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} \end{pmatrix}$$



Matrix from 5 point stencil on a 2D grid, reordered with nested dissection

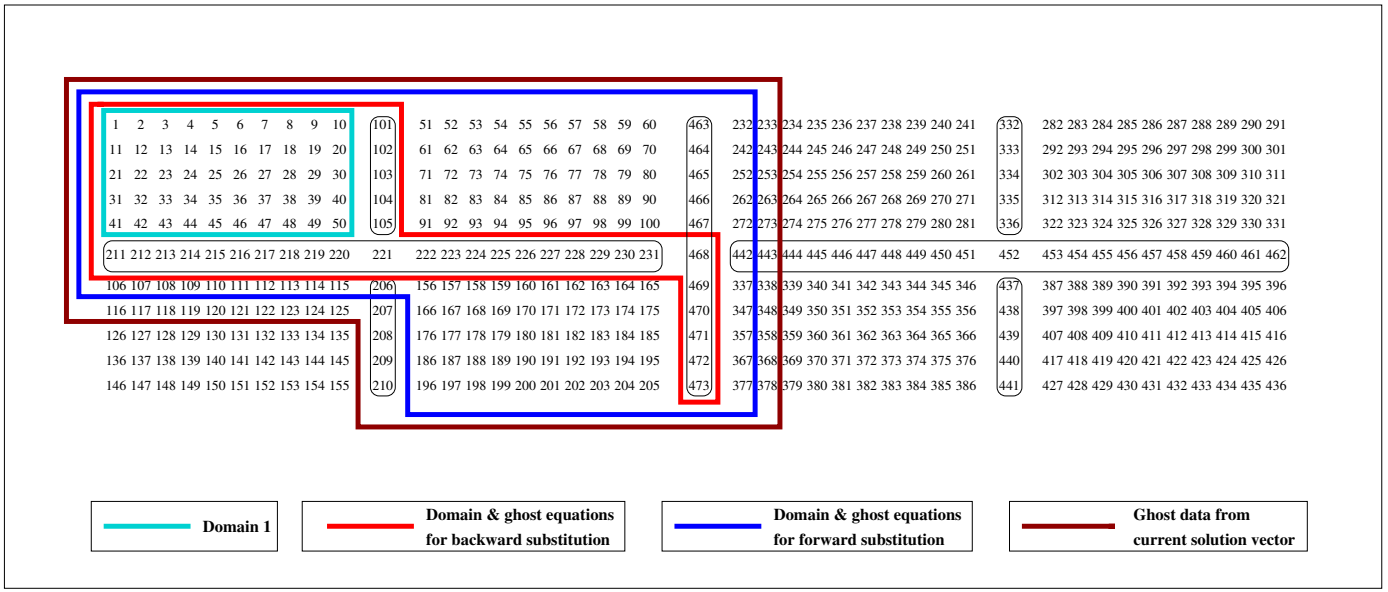
Avoid communication through ghosting

Input: $G(A)$, $G(L)$, $G(U)$,
s, number of steps; α_0, subset of unknowns
Output: Sets β_j, γ_j and δ_j for all $j = 1$ till s
for $i = 1$ *to* s
Find $\beta_i = \mathit{ReachableVertices}(G(U), \alpha_{i-1})$
Find $\gamma_i = \mathit{ReachableVertices}(G(L), \beta_i)$
Find $\delta_i = \mathit{Adj}(G(A), \gamma_i)$
Set $\alpha_i = \delta_i$
end for

Ghost data required for $i = 1 : s$

$$x(\delta_i), A(\gamma_i, \delta_i)$$

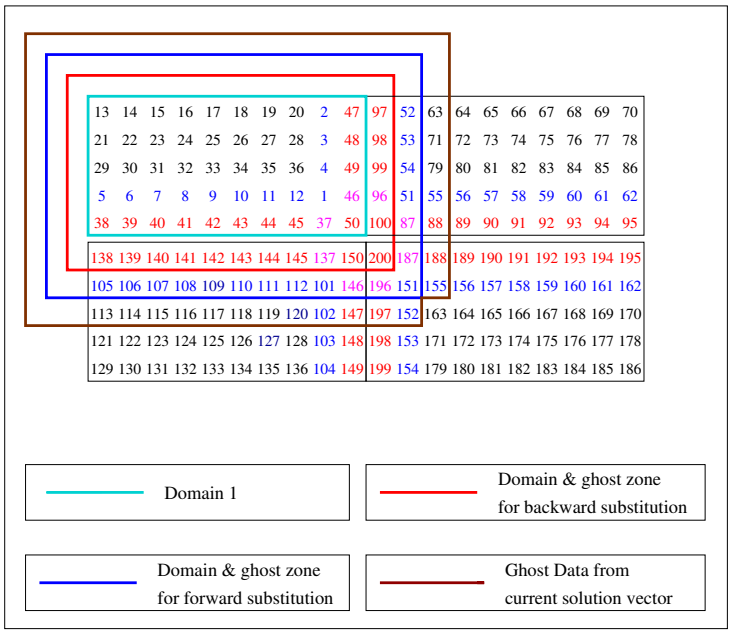
$$L(\gamma_i, \gamma_i), U(\beta_i, \beta_i)$$



⇒ Ghosting not sufficient, one processor does half of the work !

CA-ILU0 with AMML(s) reordering and ghosting

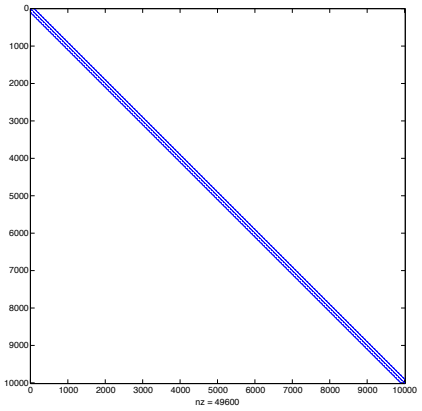
- Reduce volume of ghost data by using Alternating Min-Max Layers (AMML) reordering:
  - First number the vertices at odd distance from the separators,
  - then number the vertices at even distance from the separators.
- No communication required during the construction and the application of CA-ILU0 [Grigori and Moufawad, 2014].

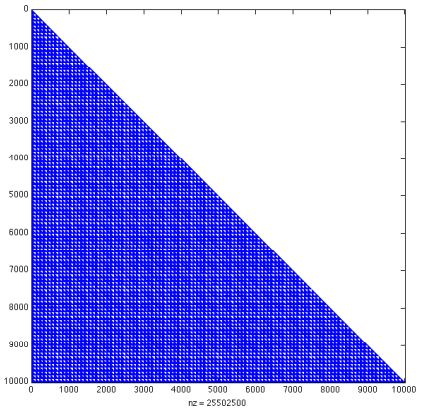


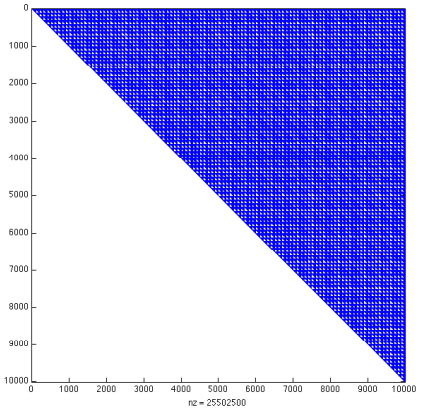
5 point stencil on a 2D grid, nested dissection + AMML(1)

Effect on the inverse of L and U

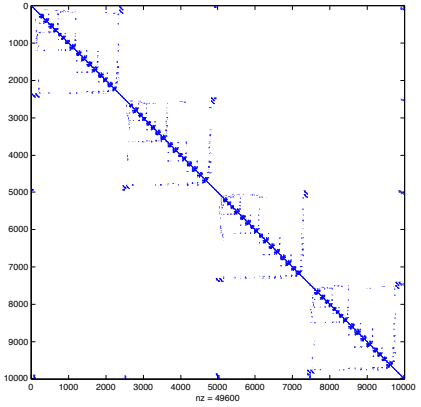
Matrix A in natural order and its L^{-1} and U^{-1} factors

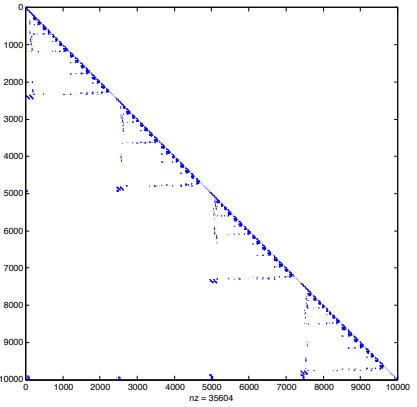


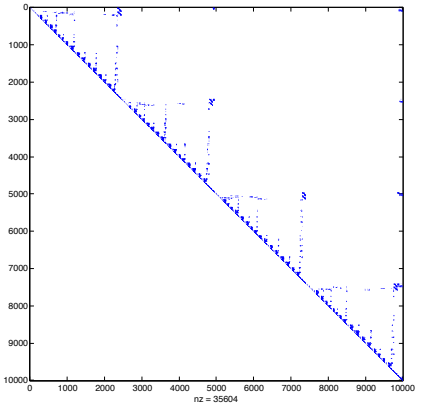




Matrix A with nested dissection and AMML(1) and its L^{-1} and U^{-1} factors







Comparison with block Jacobi

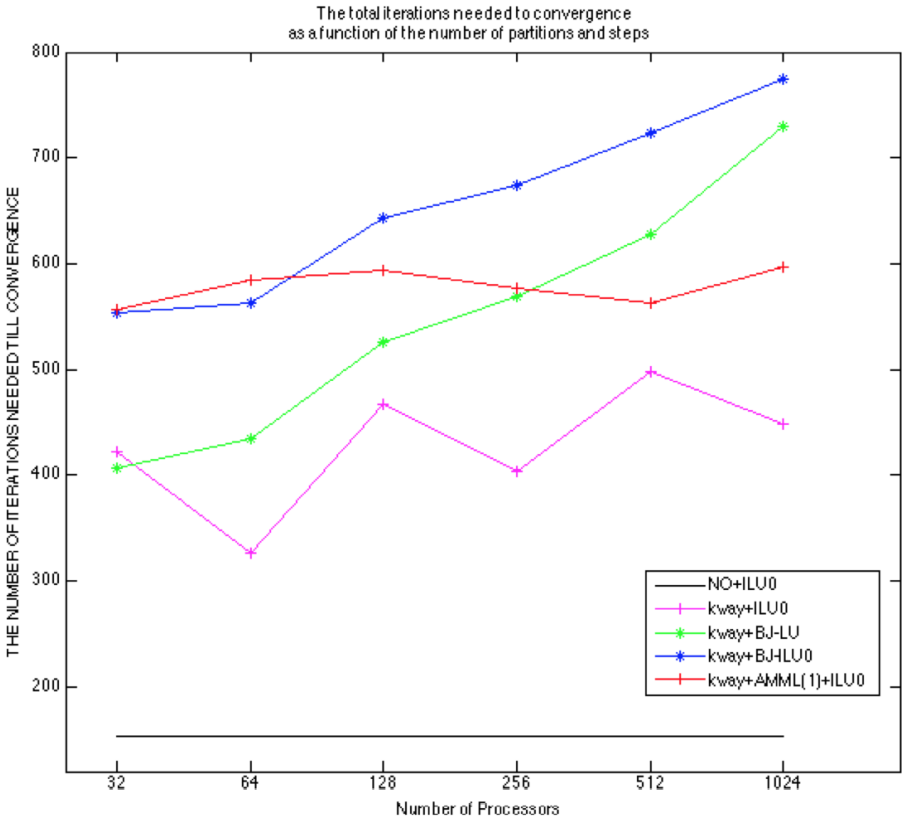
Tests for a boundary value problem (Achdou, Nataf), $40 \times 40 \times 40$ grid

3D Skyscraper Problem - SKY3D

$$
\begin{aligned}
-div(\kappa(x)\nabla u) &= f \text{ in } \Omega \\
u &= 0 \text{ on } \partial\Omega_D \\
\frac{\partial u}{\partial n} &= 0 \text{ on } \partial\Omega_N
\end{aligned}
$$

Methods tested:

- Natural ordering NO+ILU0
- CA-ILU0 - kway+AMML(1)+ILU0
- Block Jacobi using LU - BJ+ILU0
- Block Jacobi using ILU0 - BJ-ILU0



Experimental results

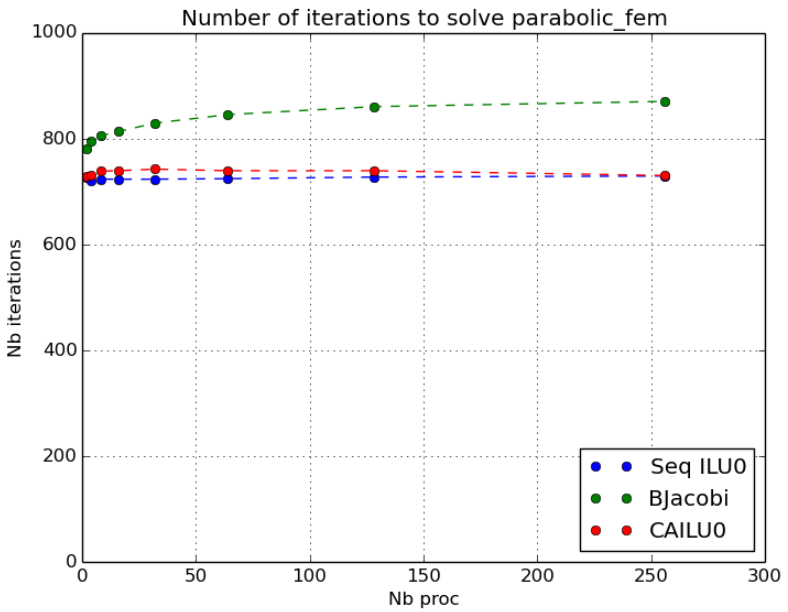


Figure : No of iterations for CA-ILU0 and block Jacobi.

Source: S. Cayrols

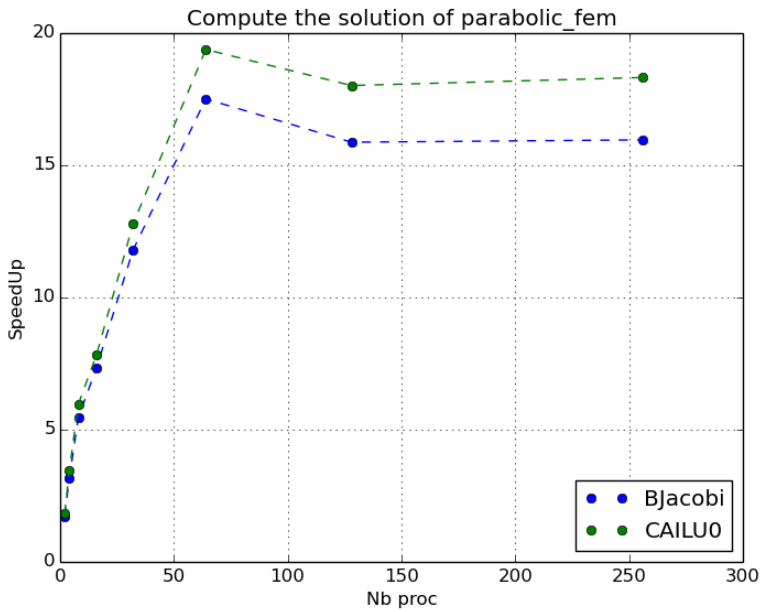


Figure : Speedup with respect to ILU0 from PETSc

References (1)

Demmel, J., Hoemmen, M., Mohiyuddin, M., and Yelick, K. (2009).

Minimizing communication in sparse matrix solvers.
In *Proceedings of the ACM/IEEE Supercomputing SC9 Conference*.

Grigori, L. and Moufawad, S. (2014).

Communication avoiding incomplete LU0 factorization.
SIAM Journal on Scientific Computing, in press.
Also as INRIA TR 8266.

Grigori, L., Moufawad, S., and Nataf, F. (2014).

Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication.
Technical Report 8597, INRIA.

Grigori, L., Stompor, R., and Szydlarski, M. (2012).

A parallel two-level preconditioner for cosmic microwave background map-making.
Proceedings of the ACM/IEEE Supercomputing SC12 Conference.

Qu, L., Grigori, L., and Nataf, F. (2013).

Parallel design and performance of nested filtering factorization preconditioner.
In *Proceedings of the ACM/IEEE Supercomputing SC12 Conference*.

Szydlarski, M., Grigori, L., and Stompor, R. (2014).

Accelerating the cosmic microwave background map-making problem through preconditioning.
Astronomy and Astrophysics Journal, Section Numerical methods and codes, 572.

Tang, J. M., Nabben, R., Vuik, C., and Erlangga, Y. A. (2009).

Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods.
J. Sci. Comput., 39:340–370.