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: $r_k \perp \mathcal{L}_k$

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

where

- $x_0$ is the initial iterate, $r_0$ is the initial residual,
- $\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\}$ is the Krylov subspace of dimension $k$,
- $\mathcal{L}_k$ 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 $\mathcal{L}_k = \mathcal{K}_k(A, r_0)$.
- Finds $x^* = A^{-1}b$ by minimizing the quadratic function

$$\phi(x) = \frac{1}{2} (x)^t Ax - b^t x$$

$$\nabla \phi(x) = Ax - b = 0$$

- 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

\[
\begin{align*}
    p_1 &= r_0 = - \nabla \phi(x_0) \\
    p_k &= r_{k-1} + \beta_k p_{k-1}
\end{align*}
\] (1)

- At k-th iteration,

\[
x_k = x_{k-1} + \alpha_k p_k = \arg\min_{x \in x_0 + K_k(A,r_0)} \phi(x)
\]

where \(\alpha_k\) is the step along \(p_k\).

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

\[
\begin{align*}
    r_k^T r_i &= 0, \text{ for all } i \neq k, \\
    p_k^T A p_i &= 0, \text{ for all } i \neq k.
\end{align*}
\]
Algorithm 1 The CG Algorithm

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

\[
\mathcal{K}_{k+1}(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^k r_0\},
\]

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

Typically, each iteration requires
- Sparse matrix vector product → point-to-point communication
- Dot products for orthogonalization → global communication
A Krylov solver finds $x_{k+1}$ from $x_0 + \mathcal{K}_{k+1}(A, r_0)$ where

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

such that the Petrov-Galerkin condition $b - Ax_{k+1} \perp \mathcal{L}_{k+1}$ is satisfied.

- Does a sequence of $k$ SpMVs to get vectors $[x_1, \ldots, x_k]$
- Finds best solution $x_{k+1}$ as linear combination of $[x_1, \ldots, x_k]$

Typically, each iteration requires

- Sparse matrix vector product $\rightarrow$ point-to-point communication
- Dot products for orthogonalization $\rightarrow$ global communication
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/~dемmel/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
Matrix-vector multiply kernel: $y(i) \leftarrow y(i) + A(i,j) \times x(j)$

for each row $i$

for $k = \text{ptr}[i]$ to $\text{ptr}[i+1]-1$ do

$y[i] = y[i] + \text{val}[k] \times x[\text{ind}[k]]$
Example: The Difficulty of Tuning

- \( n = 21200 \)
- \( \text{nnz} = 1.5 \text{ M} \)
- \( \text{kernel: SpMV} \)
- \( \text{Source: FEM discretization} \)
  \( \text{NASA structural analysis problem} \)
Example: The Difficulty of Tuning

- $n = 21200$
- $\text{nnz} = 1.5 \text{ M}$
- kernel: SpMV
- Source: NASA structural analysis problem
- $8 \times 8$ 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.

<table>
<thead>
<tr>
<th>Row Block Size (r)</th>
<th>Column Block Size (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

Best: 4x2

Reference: Matrix #02-raefsky3.rua on Itanium 2 (900 MHz) [Ref=274.3 Mflop/s]
Register Profile: Itanium 2

Example of off-line tuning: dense matrix
Another example of tuning challenges

- More complicated non-zero structure in general
- \( N = 16614 \)
- \( \text{NNZ} = 1.1M \)
- FEM fluid flow application
• More complicated non-zero structure in general

• $N = 16614$
• $\text{NNZ} = 1.1\text{M}$
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 $r \times c$ block size
  - **Off-line benchmark**
    - Precompute $Mflops(r,c)$ using dense $A$ for each $r \times c$
    - Once per machine/architecture
  - **Run-time “search”**
    - Sample $A$ to estimate $Fill(r,c)$ for each $r \times c$
  - **Run-time heuristic model**
    - Choose $r, c$ to minimize $\text{time} \sim \frac{Fill(r,c)}{Mflops(r,c)}$
Accurate and Efficient Adaptive Fill Estimation

• Idea: Sample matrix
  – Fraction of matrix to sample: $s \in [0,1]$
  – Cost $\sim O(s \times \text{nnz})$
  – 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 SpMV
  – Heuristic: 1 to 11 SpMV
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)
Accuracy of the Tuning Heuristics [Itanium 2]
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
  - \(A \cdot A^T \cdot x, A^T \cdot A \cdot x:\) \(4x\) over CSR, \(1.8x\) over RB
  - \(A^2 \cdot x:\) \(2x\) 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

20 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)**
- 4MB L2 cache
- 10.66 GB/s FSB
- 21.33 GB/s MCH
- 8 x 667MHz FB-DIMMs

**AMD Opteron 2356 (Barcelona)**
- 4MB L2 cache
- 10.66 GB/s FSB
- 21.33 GB/s MCH
- 8 x 667MHz FB-DIMMs

**Sun T2+ T5140 (Victoria Falls)**
- 4MB Shared L2 (16 way)
- 2MB victim
- 10.66 GB/s FSB
- 21.33 GB/s MCH
- 8 x 667MHz FB-DIMMs

**IBM QS20 Cell Blade**
- 512K L2 cache
- 25.6 GB/s XDR memory controllers
- 8 x 667MHz FB-DIMMs

HW switches automatically from thread waiting for memory to another

Source: Sam Williams
Multicore SMPs Used (threads)

Intel Xeon E5345 (Clovertown)
- 8 threads

AMD Opteron 2356 (Barcelona)
- 8 threads

Sun T2+ T5140 (Victoria Falls)
- 128 threads

IBM QS20 Cell Blade
- 16* threads

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
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

2K x 2K Dense matrix stored in sparse format

Well Structured (sorted by nonzeros/row)

- Protein
- FEM / Spheres
- FEM / Cantilever
- Wind Tunnel
- FEM / Harbor
- QCD
- FEM / Ship
- Economics
- Epidemiology

Poorly Structured hodgepodge

- FEM / Accelerator
- Circuit
- webbase

Extreme Aspect Ratio (linear programming)

LP

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 $x$
• 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?

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 literature, 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 $\text{span}\{b, Ab, \ldots, A^k b\}$ minimizing $||Ax - b||_2$

Cost of $k$ steps of standard GMRES vs new GMRES

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

Sequential: #words\_moved =
    $O(k \cdot \text{nnz})$ from SpMV
    $+ O(k^2 \cdot n)$ from MGS

Parallel: #messages =
    $O(k)$ from SpMV
    $+ O(k^2 \cdot \log p)$ from MGS

Slide source: J. Demmel
CA-GMRES

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

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

Sequential: \#words\_moved =
   \( O(k \cdot \text{nnz}) \) from SpMV
   + \( O(k^2 \cdot n) \) from MGS
Parallel: \#messages =
   \( O(k) \) from SpMV
   + \( O(k^2 \cdot \log p) \) from MGS

Communication-avoiding GMRES
\( W = [ v, Av, A^2 v, \ldots, A^k v ] \)
\( [Q,R] = \text{TSQR}(W) \) "Tall Skinny QR"
Build \( H \) from \( R \), solve LSQ problem

Sequential: \#words\_moved =
   \( O(\text{nnz}) \) from SpMV
   + \( O(k \cdot n) \) from TSQR
Parallel: \#messages =
   \( O(1) \) from computing \( W \)
   + \( O(\log p) \) from TSQR

Slide source: J. Demmel

17 of 54
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.

\[
Ax = \begin{pmatrix}
* & * & * \\
* & * & * & * \\
* & * & * & * & * \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\end{pmatrix}
\begin{pmatrix}
* \\
* \\
* \\
\vdots \\
\end{pmatrix}
= \begin{pmatrix}
* \\
* \\
* \\
\vdots \\
\end{pmatrix}
\]
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

\[
Ax = \begin{pmatrix}
* & * & & \\
* & * & * & & \\
* & & * & * & \\
& & & & \\
\end{pmatrix} \cdot \begin{pmatrix}
* \\
* \\
* \\
\end{pmatrix} = \begin{pmatrix}
* \\
* \\
* \\
\end{pmatrix}
\]
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

\[
Ax = \begin{pmatrix}
* & * & & & & \\
* & * & * & & & \\
* & * & * & * & & \\
& & & \ddots & & \\
& & & & & \\
& & & & & \\
\end{pmatrix}
\begin{pmatrix}
* \\
* \\
* \\
* \\
* \\
* \\
\end{pmatrix} = 
\begin{pmatrix}
* \\
* \\
* \\
* \\
* \\
* \\
\end{pmatrix}
\]
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

\[
Ax = \begin{pmatrix}
* & * & * \\
* & * & * & * \\
* & * & * & * & * \\
& * & * & * & * & * \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
\end{pmatrix} \begin{pmatrix}
* \\
* \\
* \\
\vdots \\
\end{pmatrix} = \begin{pmatrix}
* \\
* \\
* \\
\vdots \\
\end{pmatrix}
\]
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

\[
Ax = \begin{pmatrix}
* & * & & \\
* & * & * & \\
& * & * & * \\
& & & \\
& & & \\
& & & \\
\end{pmatrix}
\begin{pmatrix}
* \\
* \\
* \\
\vdots \\
\vdots \\
\end{pmatrix}
= \begin{pmatrix}
* \\
* \\
* \\
\vdots \\
\vdots \\
\end{pmatrix}
\]

\[
A^3x, \quad A^2x, \quad Ax, \quad x
\]

\[
1, \quad 2, \quad 3, \quad 4, \ldots
\]

\[
\ldots, \quad 32
\]
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

\[
Ax = \begin{pmatrix}
* & * \\
* & * & * \\
* & * & * & * \\
\vdots & \vdots & \vdots & \vdots
\end{pmatrix}
\begin{pmatrix}
* \\
* \\
* \\
\vdots
\end{pmatrix} = \begin{pmatrix}
* \\
* \\
* \\
\vdots
\end{pmatrix}
\]
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)

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60

- Matrix powers
- Kernel
- TSQR
- Block Gram-Schmidt
- Small dense operations
- Sparse matrix-vector product
- Modified
- Gram-Schmidt

Relative runtime, for best (k,t)

Sparse matrix name:
- pwtk
- bmw
- xenon
- cant
- 1d3pt
- cfd
- shipsec

With floor(restart length / k) = t

k=5
- 2.3x
- 2.1x
- 1.7x
- 2.1x
- 4.3x
- 1.7x
- 1.6x
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,
    
    $r_{k-1} \rightarrow T(r_{k-1}) = \begin{bmatrix}
    * & 0 & 0 \\
    \vdots & \ddots & \vdots \\
    0 & * & 0 \\
    \vdots & \ddots & \ddots \\
    0 & 0 & * \\
    \vdots & \ddots & \ddots \\
    0 & 0 & 0
    \end{bmatrix}$, $T_s(r_{k-1}) = \{ T(r_{k-1})(:, 1), \ldots, T(r_{k-1})(:, t) \}$
  - generate $t$ new basis vectors, obtain an enlarged Krylov subspace
    
    $\mathcal{H}_{t,k}(A, r_0) = \text{span}\{ T_s(r_0), A T_s(r_0), A^2 T_s(r_0), \ldots, A^{k-1} T_s(r_0) \}$
  - search for the solution of the system $Ax = b$ in $\mathcal{H}_{t,k}(A, r_0)$
Properties of enlarged Krylov subspaces

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

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

- For all $k < k_{\max}$ the dimensions of $\mathcal{H}_{t,k}$ and $\mathcal{H}_{t,k+1}$ are strictly 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.

$$\mathcal{H}_{t,1}(A, r_0) \subset \ldots \subset \mathcal{H}_{t,k_{\max}-1}(A, r_0) \subset \mathcal{H}_{t,k_{\max}}(A, r_0) = \mathcal{H}_{t,k_{\max}+q}(A, r_0), \forall q > 0$$
Let $\mathcal{K}_{p_{\max}} = \mathcal{K}_{p_{\max}} + q$ and $\mathcal{H}_{t,k_{\max}} = \mathcal{H}_{t,k_{\max}} + q$ for $q > 0$. Then

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

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

Enlarged Krylov subspace methods based on CG

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

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

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

$$\phi(x_k) = \min\{\phi(x), \forall x \in x_0 + \mathcal{H}_{t,k}(A, r_0)\}$$
Convergence analysis

**Given**
- $A$ is an SPD matrix, $x^*$ is the solution of $Ax = b$
- $\|e_k\|_A = \|x^* - x_k\|_A$ 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 $\bar{K}$ iterations

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

\[ \|e_k\|_A = \|x^* - x_k\|_A \leq \|e_k\|_A \]
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 \( \mathcal{K}_{t,k} \)
- At each \( k^{th} \) iteration, approximate the solution as
  \[
  x_k = x_{k-1} + Q_k \alpha_k
  \]
  such that
  \[
  \phi(x_k) = \min\{\phi(x), \forall x \in x_0 + \mathcal{K}_{t,k}\}
  \]
- 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.
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:

\[
\alpha_k = W_k^t r_{k-1}, \\
x_k = x_{k-1} + W_k \alpha_k, \\
r_k = r_{k-1} - AW_k \alpha_k,
\]

$W_k$ needs to be A-orthormalized only against $W_{k-1}$ and $W_{k-2}$. 
Algorithm 2 The SRE-CG algorithm

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

1: $r_0 = b - Ax_0$, $\rho_0 = \|r_0\|^2_2$, $k = 1$
2: while $(\sqrt{\rho_{k-1}} > \epsilon \|b\|_2$ and $k < k_{\text{max}}$) do
3: if $k = 1$ then
4: Let $W_1 = T(r_0)$, A-orthonormalise its vectors
5: else
6: Let $W_k = AW_{k-1}$
7: A-orthonormalise $W_k$ against $W_{k-1}$ and $W_{k-2}$ if $k > 2$
8: A-orthonormalise the vectors of $W_k$
9: end if
10: $\alpha_k = (W_k^t r_{k-1})$
11: $x_k = x_{k-1} + W_k \alpha_k$
12: $r_k = r_{k-1} - AW_k \alpha_k$
13: $\rho_k = \|r_k\|^2_2$
14: $k = k + 1$
15: end while
SRE-CG: cost on \( t \) processors

Cost of \( \bar{k} \) iterations of CG is:

- Total Flops \( \approx 2\text{nnz} \cdot \bar{k}/t + 4n\bar{k}/t \)
- # words \( \approx O(\bar{k}) \) (from SpMV)
- # messages \( \approx 2k \log(t) + O(k) \) (from SpMV)

Cost of \( k \) iterations of SRE-CG is:

- Total Flops \( \approx 2\text{nnz} \cdot k + O(ntk) \)
- # words \( \approx kt^2\log(t) + O(k) \) (from SpMV)
- # messages \( \approx k\log(t) + O(k) \) (from SpMV)

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

<table>
<thead>
<tr>
<th>Pa</th>
<th>SKY3D</th>
<th>ANI3D</th>
<th>ELAST3D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CG</td>
<td>SRE-CG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iter</td>
<td>Iter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Err</td>
<td>Err</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>902</td>
<td>211</td>
<td>1E-5</td>
</tr>
<tr>
<td>16</td>
<td>902</td>
<td>119</td>
<td>9E-6</td>
</tr>
<tr>
<td>32</td>
<td>902</td>
<td>43</td>
<td>4E-6</td>
</tr>
<tr>
<td>2</td>
<td>4187</td>
<td>875</td>
<td>7E-5</td>
</tr>
<tr>
<td>4</td>
<td>4146</td>
<td>673</td>
<td>8E-5</td>
</tr>
<tr>
<td>8</td>
<td>4146</td>
<td>449</td>
<td>1E-4</td>
</tr>
<tr>
<td>16</td>
<td>4146</td>
<td>253</td>
<td>2E-4</td>
</tr>
<tr>
<td>32</td>
<td>4146</td>
<td>148</td>
<td>2E-4</td>
</tr>
<tr>
<td>64</td>
<td>4146</td>
<td>92</td>
<td>1E-4</td>
</tr>
<tr>
<td>2</td>
<td>1098</td>
<td>652</td>
<td>1E-7</td>
</tr>
<tr>
<td>4</td>
<td>1098</td>
<td>445</td>
<td>1E-7</td>
</tr>
<tr>
<td>8</td>
<td>1098</td>
<td>321</td>
<td>8E-8</td>
</tr>
<tr>
<td>16</td>
<td>1098</td>
<td>238</td>
<td>4E-8</td>
</tr>
<tr>
<td>32</td>
<td>1098</td>
<td>168</td>
<td>5E-8</td>
</tr>
<tr>
<td>64</td>
<td>1098</td>
<td>116</td>
<td>1E-8</td>
</tr>
</tbody>
</table>
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
  \[ A\mathbf{x} = b. \]

- Convergence depends on \( \kappa(A) \) and the eigenvalue distribution (for SPD matrices).

- To accelerate convergence, solve
  \[ M^{-1}A\mathbf{x} = M^{-1}b, \]
  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

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

block Jacobi preconditioner is:

$$M = \begin{pmatrix} A_{11} \\ \vdots \\ A_{PP} \end{pmatrix} = \begin{pmatrix} L_{11} U_{11} \\ \vdots \\ L_{PP} U_{PP} \end{pmatrix} = LU$$
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 analysis (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:

\[ P := I - AZE^{-1}Z^T, \quad E := Z^TAZ \]

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

\[ P_{2/lvl}^{-1} = M^{-1}P + ZE^{-1}Z^T, \]

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

- \( P_{2/lvl}^{-1} AZ = Z \)
- The small eigenvalues are shifted to 1.
- \( P_{2/lvl} \) 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 (e.g., 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 $y = ZE^{-1}Z^T(Ax_i) = ZE^{-1}Z^Tv$
  $\Rightarrow$ involves collective communication when computing $Z^Tv$, and solving a linear system with $E$. 

\[ \begin{array}{c|c|c|c}
Z & E^{-1} & Z^T & (Ax_i) \\
\hline
= & & & \\
\end{array} \]

\[ \begin{array}{c|c|c|c}
Z & E^{-1}(Z^T Ax_i) \\
\end{array} \]
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 $E$. 

![Diagram showing the computation process](image-url)
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

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 $N = \langle n_t n_T \rangle$, and $N^{-1}$ is a block diagonal symmetric Toeplitz matrix.

- The solution of the generalized least squares problem is found by solving

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

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:

\[
M_{\text{diag}} S x = M_{\text{diag}} b, \quad \text{where} \\
S := A^T N^{-1} A, \quad b := A^T N^{-1} d, \quad M_{\text{diag}} := (A^T \text{diag}(N^{-1})A)^{-1}
\]

- The diagonal preconditioner \(M_{\text{diag}}\) does not scale numerically.

Figure: Eigenvalue distribution of \(S\) and \(M_{\text{diag}}^{-1} S\) (NoM and \(M_{\text{diag}}\) 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
  \[ M_{2lvl} = M_{\text{diag}}(I - S(ZE^{-1}Z^T)) + ZE^{-1}Z^T, \]
  where \( M_{\text{diag}} = (A^T \text{diag}(N^{-1})A)^{-1}, \ E = Z^T SZ \)

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

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

block Jacobi preconditioner is:

$$M = \begin{pmatrix}
A_{11} \\
\vdots \\
A_{PP}
\end{pmatrix} = \begin{pmatrix}
L_{11} U_{11} \\
\vdots \\
L_{PP} U_{PP}
\end{pmatrix} = LU$$
A preconditioned matrix powers kernel computes the set basis vectors
\[
\{ M^{-1} y_0, (M^{-1} A)^2 y_0, \ldots, (M^{-1} A)^{s-1} y_0, (M^{-1} A)^s y_0 \}
\]
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 $y_i = (LU)^{-1} A y_{i-1}$ as:

1. Compute $f = A y_{i-1}$
2. Solve $LU y_i = f$ i.e.
   2.1 Solve $L z = f$ by forward substitution
   2.2 Solve $U y_i = z$ by backward substitution
Can we compute $s$ iterations with no communication?

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

1. Compute $f = Ay_{i-1}$
2. Solve $Lz = f$ by forward substitution
3. Solve $Uy_i = z$ by backward substitution

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; $\alpha_0$, subset of unknowns

Output: Sets $\beta_j, \gamma_j$ and $\delta_j$ for all $j = 1$ till $s$

for $i = 1$ to $s$
  Find $\beta_i = \text{ReachableVertices}(G(U), \alpha_{i-1})$
  Find $\gamma_i = \text{ReachableVertices}(G(L), \beta_i)$
  Find $\delta_i = \text{Adj}(G(A), \gamma_i)$
  Set $\alpha_i = \delta_i$
end for

Ghost data required for $i = 1 : s$

$\chi(\delta_i), A(\gamma_i, \delta_i)$
$L(\gamma_i, \gamma_i), U(\beta_i, \beta_i)$

⇒ 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

$-\text{div}(\kappa(x) \nabla u) = f \text{ in } \Omega$

$u = 0 \text{ on } \partial \Omega_D$

$\frac{\partial u}{\partial n} = 0 \text{ on } \partial \Omega_N$

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

The total iterations needed to converge as a function of the number of processors and steps

The number of iterations needed till convergence

Number of Processors
Experimental results

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

*Source: S. Cayrols*

**Figure**: Speedup with respect to ILU0 from PETSc
Minimizing communication in sparse matrix solvers.

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

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

A parallel two-level preconditioner for cosmic microwave background map-making.

Parallel design and performance of nested filtering factorization preconditioner.

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

Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods.