Sparse linear solvers: iterative methods and preconditioning

L. Grigori

ALPINES
INRIA and LJLL, UPMC

March 2017

Plan

Sparse linear solvers
- Sparse matrices and graphs
- Classes of linear solvers

Krylov subspace methods
- Conjugate gradient method

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

Plan

Sparse matrices and graphs

- Most matrices arising from real applications are sparse.
- A 1M-by-1M submatrix of the web connectivity graph, constructed from an archive at the Stanford WebBase.

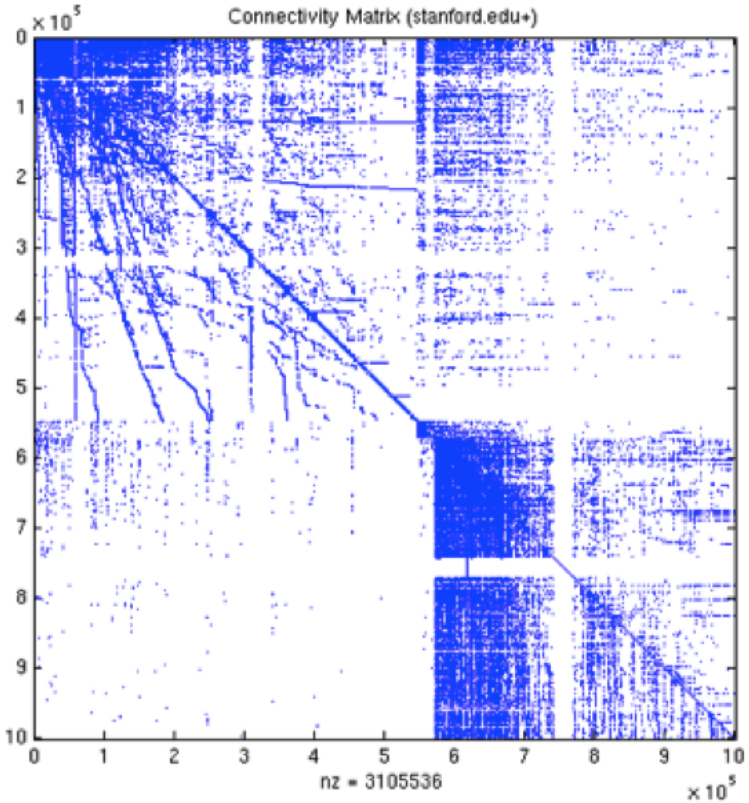


Figure : Nonzero structure of the matrix

Sparse matrices and graphs

- Most matrices arising from real applications are sparse.
- GHS class: Car surface mesh, $n = 100196, nnz(A) = 544688$

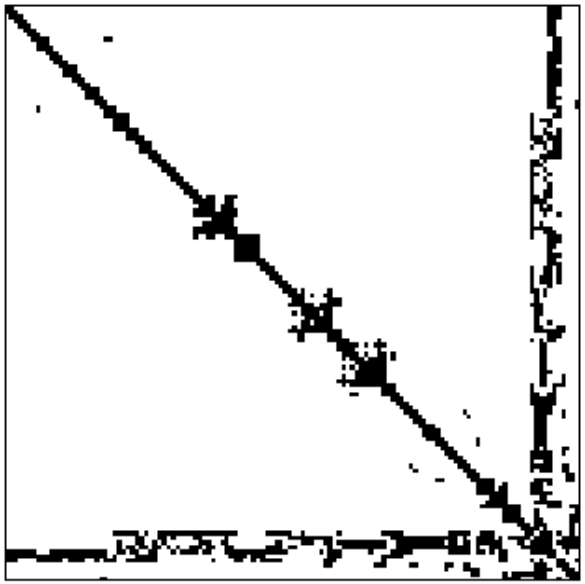


Figure : Nonzero structure of the matrix

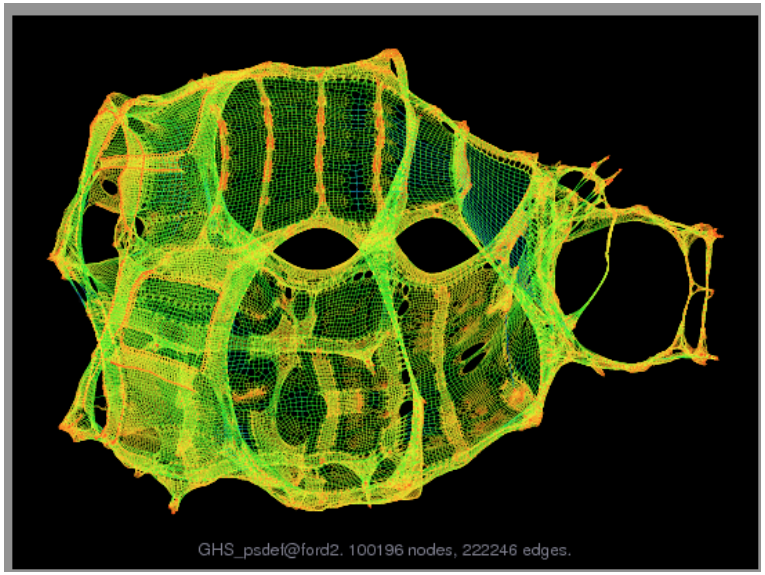


Figure : Its undirected graph

Examples from Tim Davis's Sparse Matrix Collection,
`http://www.cise.ufl.edu/research/sparse/matrices/`

Sparse matrices and graphs

- Semiconductor simulation matrix from Steve Hamm, Motorola, Inc. circuit with no parasitics, $n = 105676$, $nnz(A) = 513072$

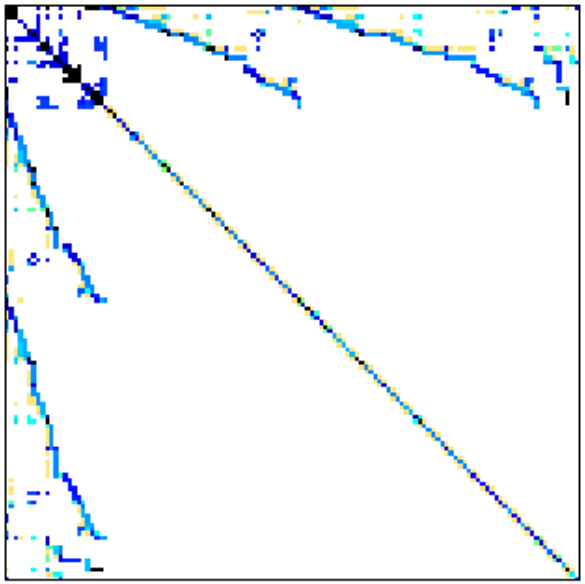


Figure : Nonzero structure of the matrix

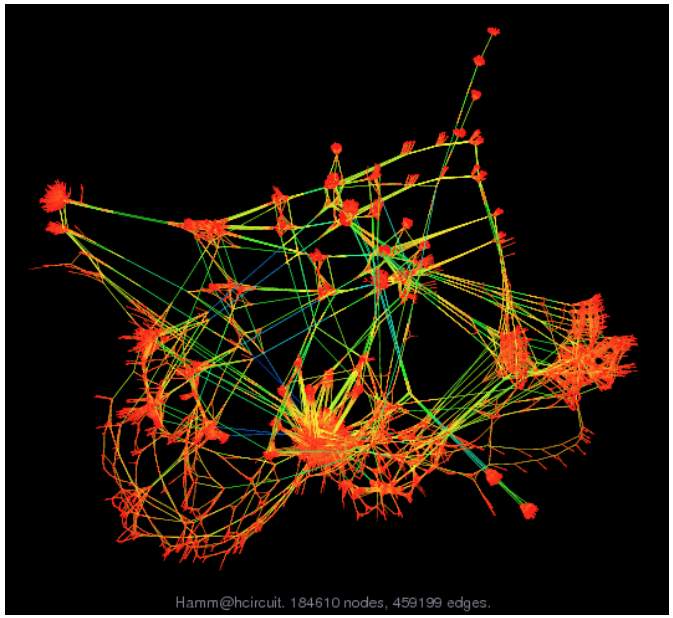


Figure : Its undirected graph

Examples from Tim Davis's Sparse Matrix Collection,
`http://www.cise.ufl.edu/research/sparse/matrices/`

Sparse linear solvers

Direct methods of factorization

- For solving $Ax = b$, least squares problems
  - Cholesky, LU, QR, LDL^T factorizations
- Limited by fill-in/memory consumption and scalability

Iterative solvers

- For solving $Ax = b$, least squares, $Ax = \lambda x$, SVD
- When only multiplying A by a vector is possible
- Limited by accuracy/convergence

Hybrid methods

As domain decomposition methods

Plan

Sparse linear solvers

Krylov subspace methods

- Conjugate gradient method

Iterative solvers that reduce communication

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.

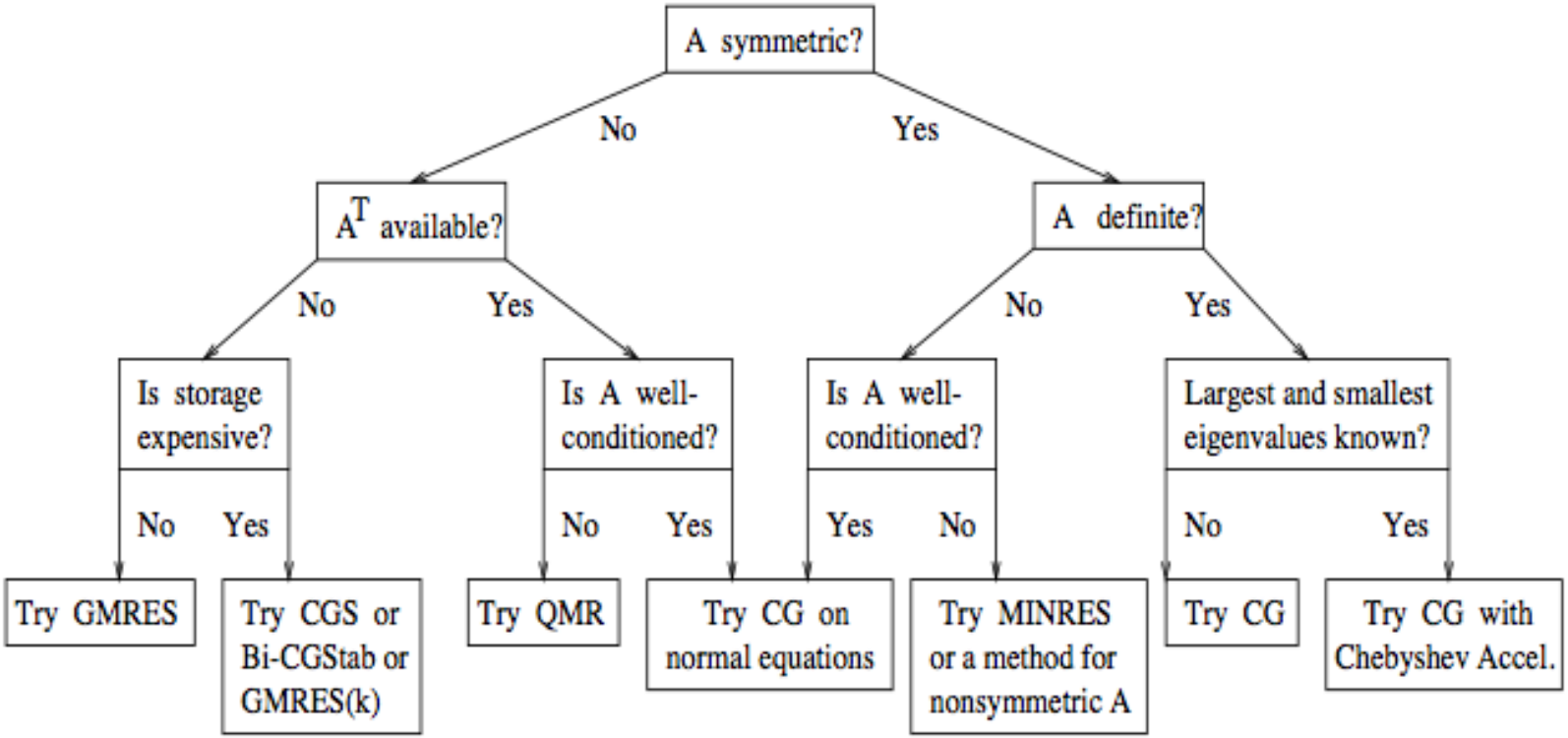
One of Top Ten Algorithms of the 20th Century

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

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

- 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

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

Challenge in getting efficient and scalable solvers

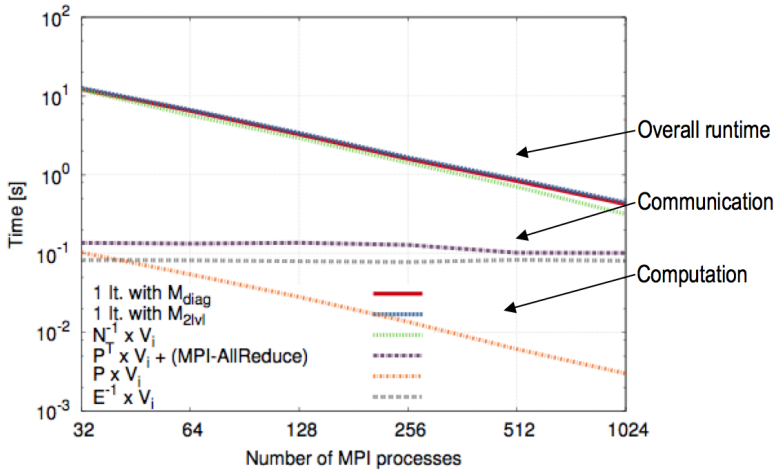
- A Krylov solver finds x_{k+1} from $x_0 + \mathcal{K}_{k+1}(A, r_0)$ where

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

  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

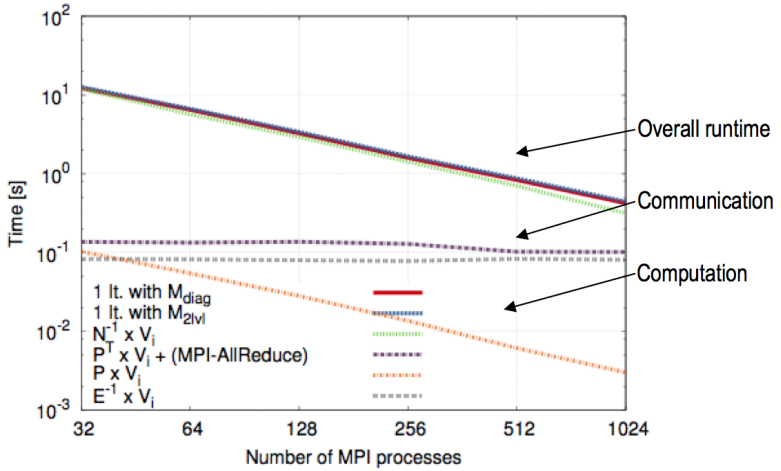
- A Krylov solver finds x_{k+1} from $x_0 + \mathcal{K}_{k+1}(A, r_0)$ where

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

  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

- Improve the performance of sparse matrix-vector product.
- Improve the performance of collective communication.
- 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

Sparse linear solvers

Krylov subspace methods

Iterative solvers that reduce communication

- CA solvers based on s-step methods
- Enlarged Krylov methods

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 communication.
- 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

Source of following 11 slides: 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

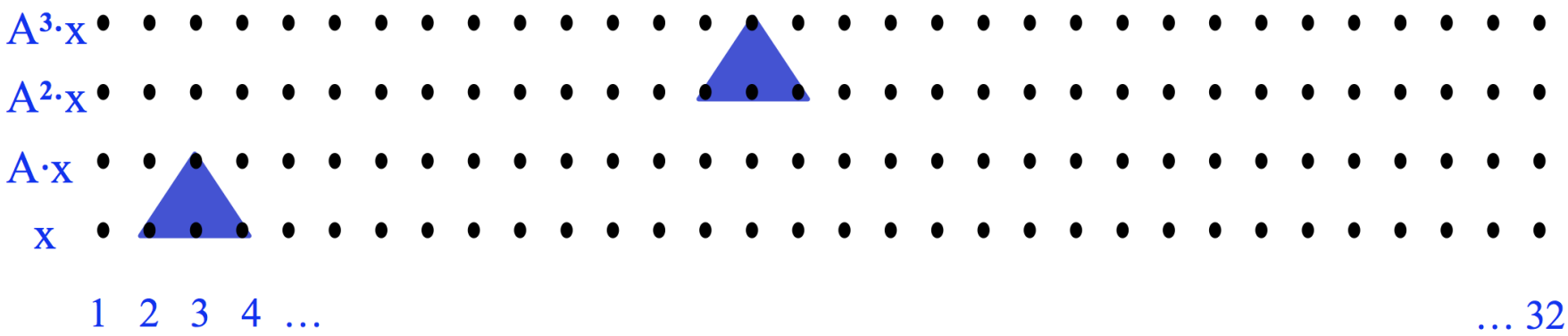
Sequential: #words_moved =
O(nnz) from SpMV
+ O(k·n) from TSQR
Parallel: #messages =
O(1) from computing W
+ O(log p) from TSQR

Source of following 11 slides: 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, $n = 32$, $k = 3$
- Shaded triangles represent data computed redundantly

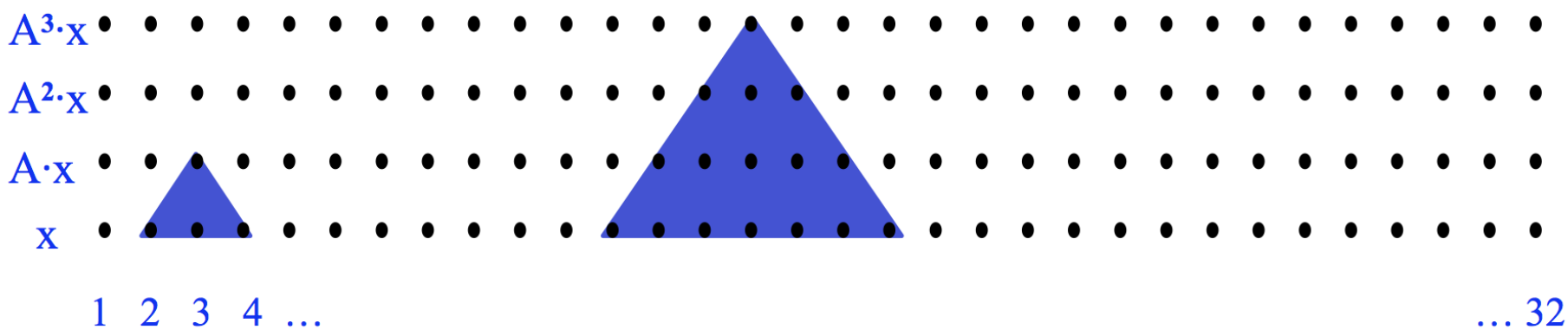
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



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, $n = 32$, $k = 3$
- Shaded triangles represent data computed redundantly

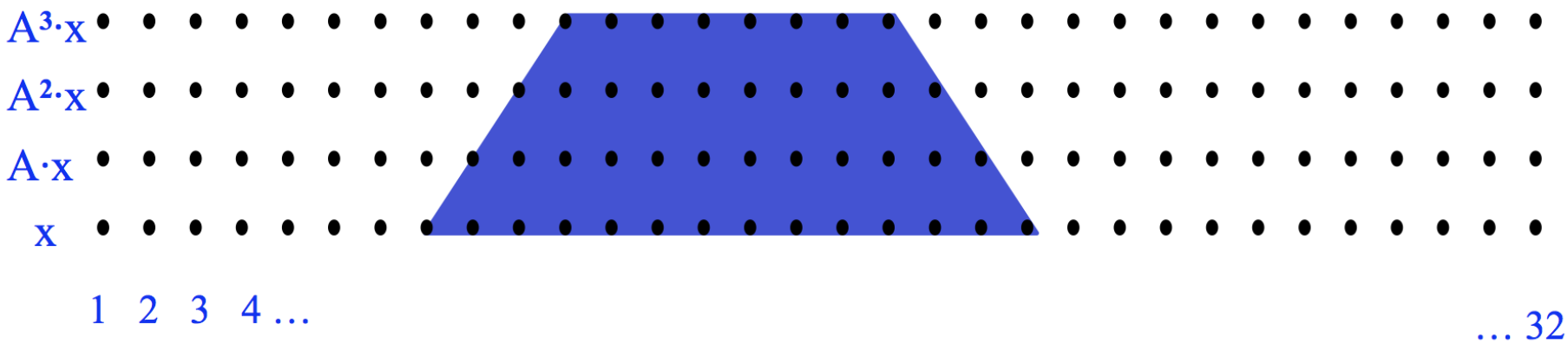
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



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, $n = 32$, $k = 3$
- Shaded triangles represent data computed redundantly

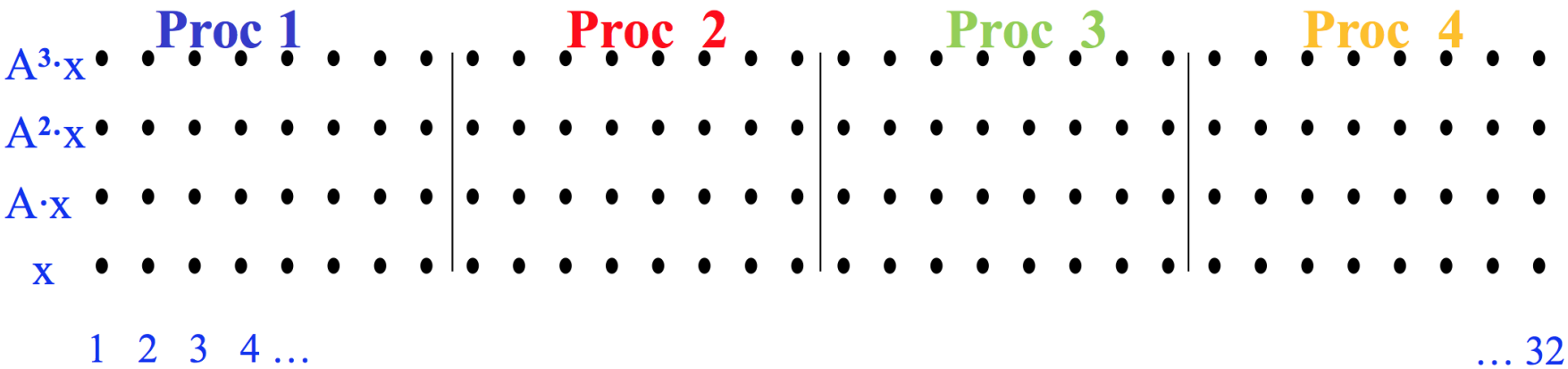
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



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, $n = 32$, $k = 3$
- Shaded triangles represent data computed redundantly

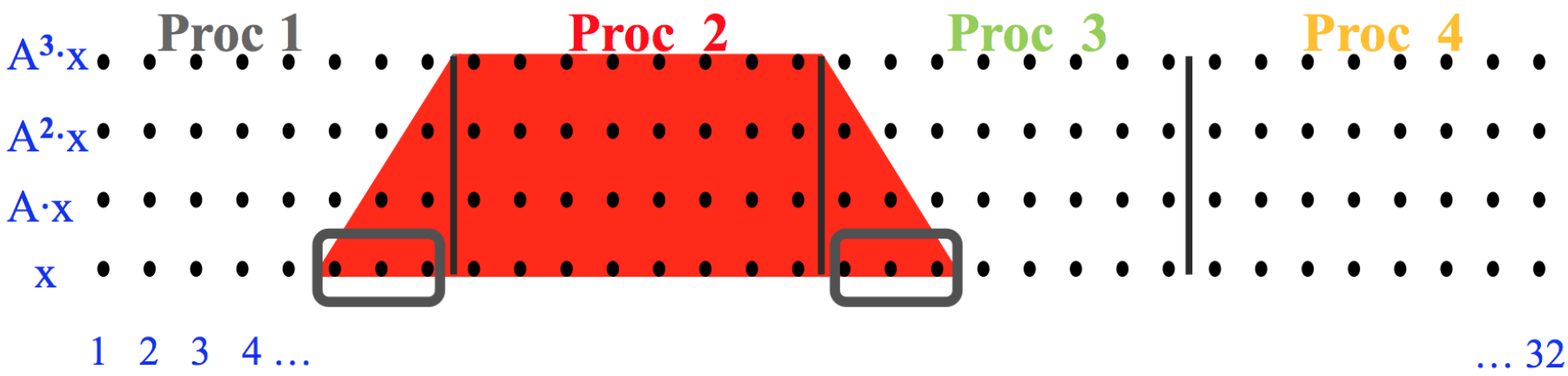
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



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, $n = 32$, $k = 3$
- Shaded triangles represent data computed redundantly

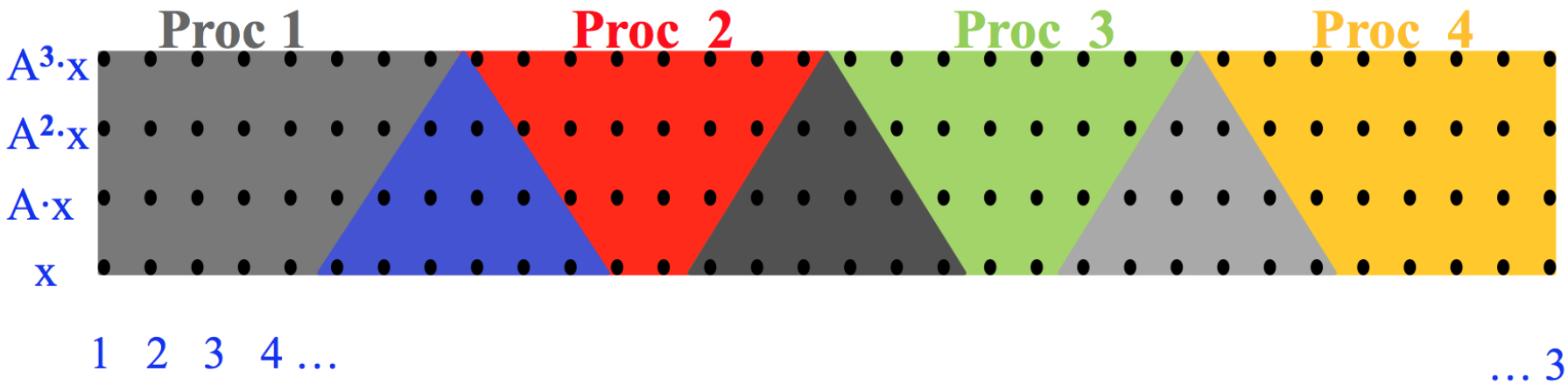
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



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, $n = 32$, $k = 3$
- Shaded triangles represent data computed redundantly

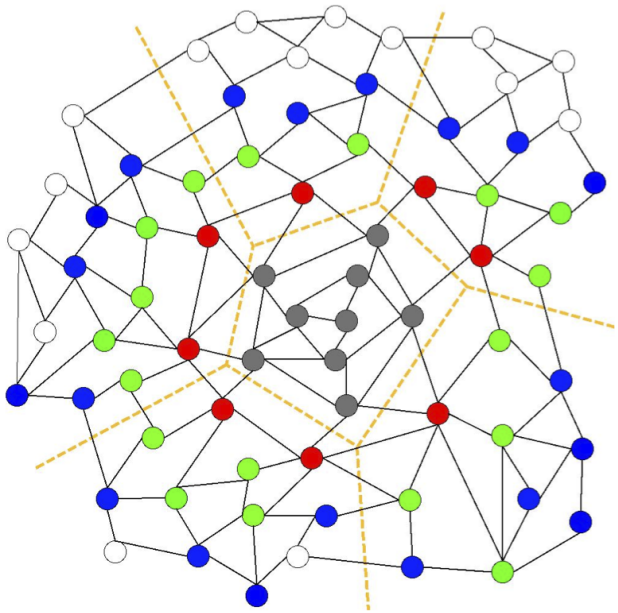
$$Ax = \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & \ddots & \ddots & \ddots & \end{pmatrix} \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ \vdots \end{pmatrix}$$



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.



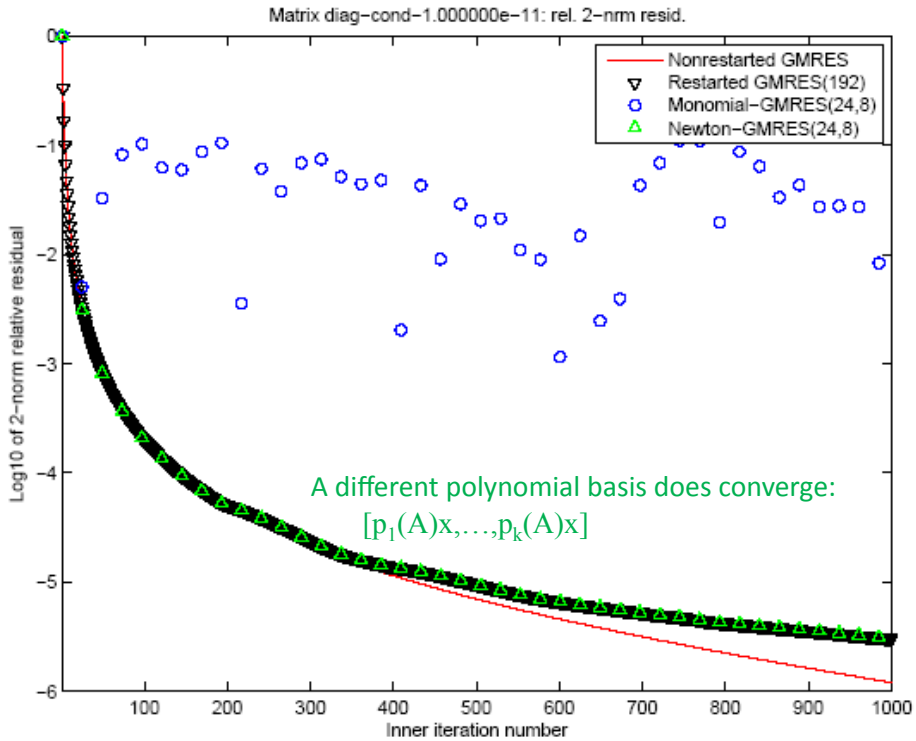
Challenges and research opportunities

Length of the basis k is limited by

- Size of ghost data
- Loss of precision

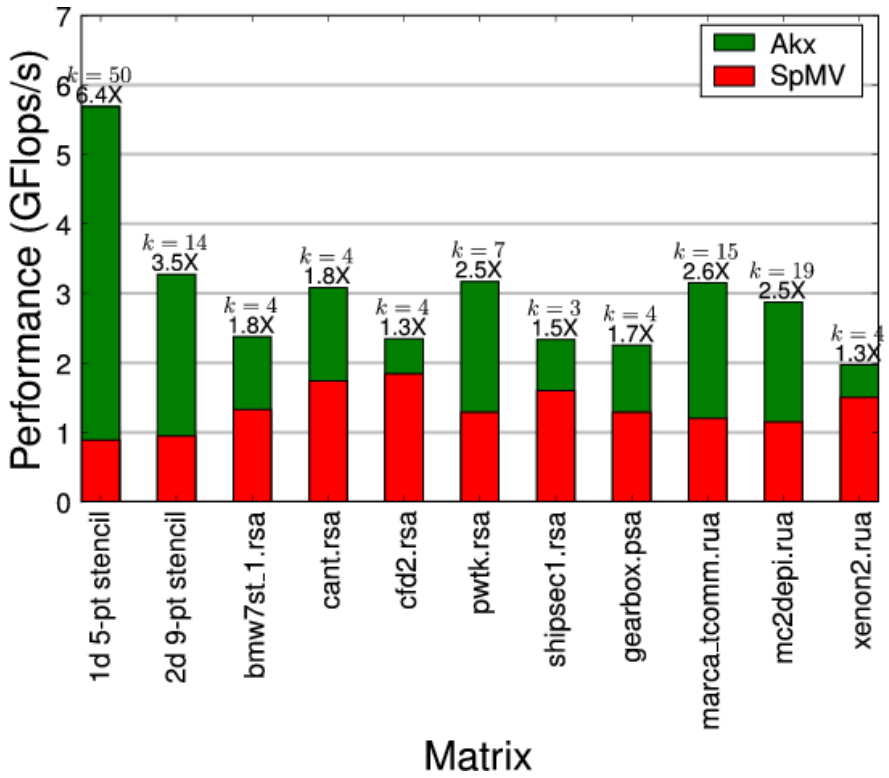
Preconditioners: lots of recent work

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

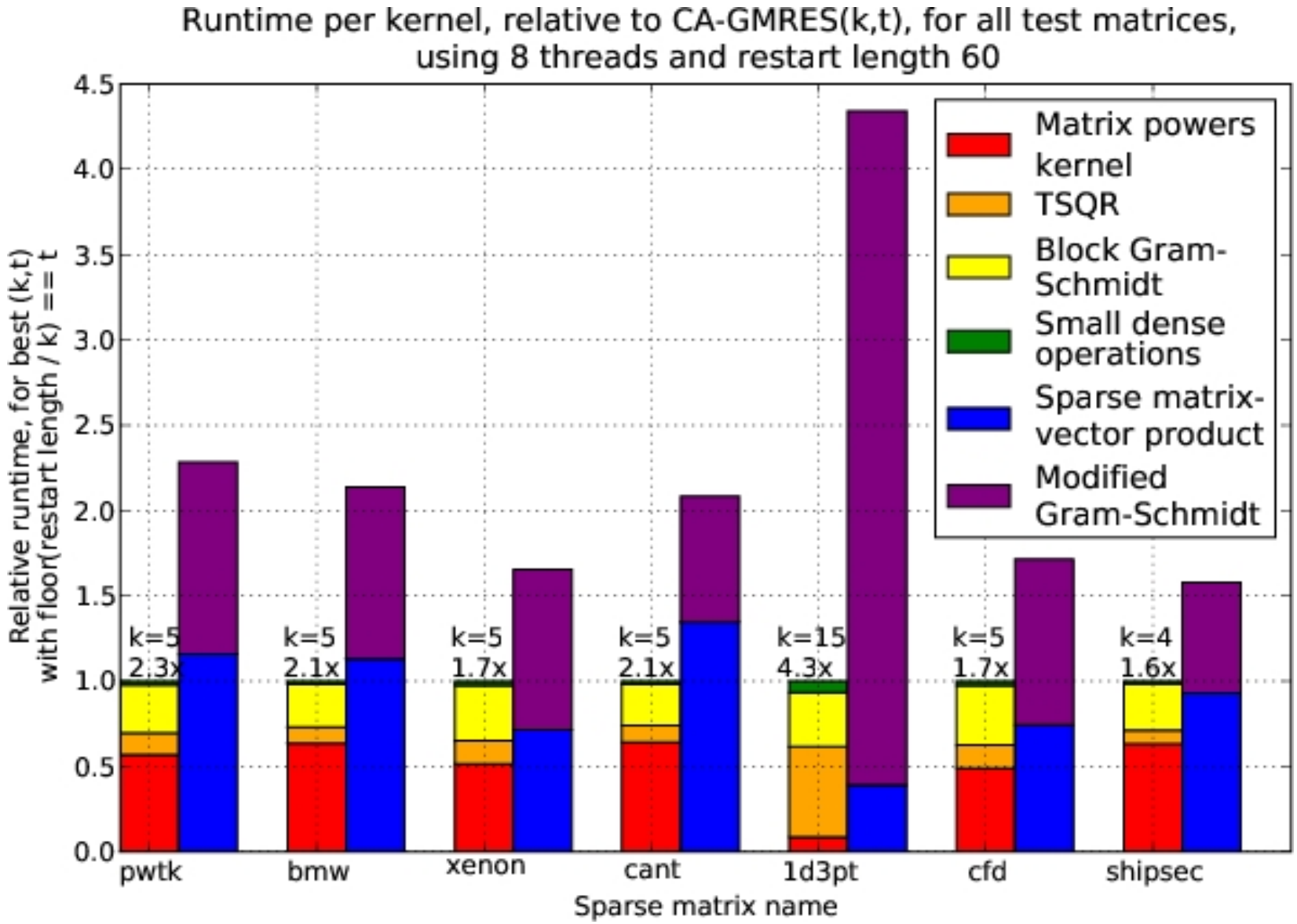


Performance

- 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)

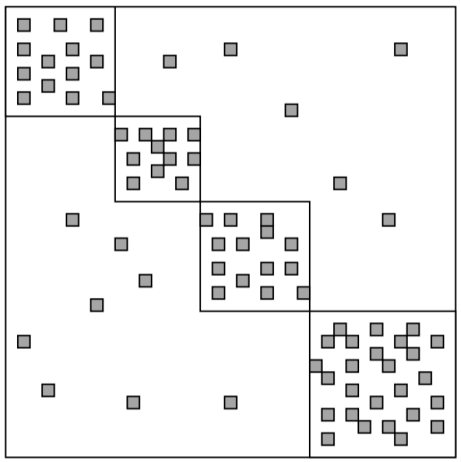


Performance (contd)



Enlarged Krylov methods [Grigori et al., 2014a]

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



$$r_0 \rightarrow T(r_0) = \begin{bmatrix} * & 0 & & 0 \\ \vdots & \vdots & & \vdots \\ * & 0 & & 0 \\ 0 & * & & 0 \\ \vdots & \vdots & & \vdots \\ 0 & * & & 0 \\ & & \ddots & \\ 0 & 0 & & * \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & * \end{bmatrix}$$

- 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.

Related work

- Block Krylov methods (O'Leary 1980): solve systems with multiple rhs

$$AX = B,$$

by searching for an approximate solution $X_k \in X_0 + \mathscr{K}_k(A, R_0)$,

$$\mathscr{K}_k(A, R_0) = block - span\{R_0, AR_0, A^2R_0, ..., A^{k-1}R_0\}.$$

- coopCG (Bhaya et al, 2012): solve one system by starting with t different initial guesses, equivalent to solving

$$AX = b * ones(1, t)$$

where X_0 is a block-vector containing the t initial guesses.

Classical CG *vs.* Enlarged CG derived from Block CG

Algorithm 3 Classic CG

$r_0 = b - Ax_0$
$p_1 = \frac{r_0}{\sqrt{r_0^t A r_0}}$
while $||r_{k-1}||_2 > \varepsilon ||b||_2$ **do**
  $\alpha_k = p_k^t r_{k-1}$
  $x_k = x_{k-1} + p_k \alpha_k$
  $r_k = r_{k-1} - A p_k \alpha_k$
  $p_{k+1} = r_k - p_k (p_k^t A r_k)$
  $p_{k+1} = \frac{p_{k+1}}{\sqrt{p_{k+1}^t A p_{k+1}}}$
end while

Algorithm 4 ECG(Odir)

$R_0 = T(b - Ax_0)$
$P_1 = \text{A-orthonormalize}(R_0)$
while $|| \sum_{i=1}^{t} R_k^{(i)} ||_2 < \varepsilon ||b||_2$ **do**
  $\alpha_k = P_k^t R_{k-1}$ ▷ $t \times t$
  $X_k = X_{k-1} + P_k \alpha_k$ ▷ $n \times t$
  $R_k = R_{k-1} - A P_k \alpha_k$ ▷ $n \times t$
  $P_{k+1} = AP_k - P_k(P_k^t AAP_k) - P_{k-1}(P_{k-1}^t AAP_k)$ ▷ $n \times t$
  $P_{k+1} = \text{A-orthonormalize}(P_{k+1})$
end while
$x = \sum_{i=1}^{t} X_k^{(i)}$ ▷ $n \times 1$

- EK-CG based on Orthodir (Lanczos formula) [Ashby et al., 1990]
- More stable than Orthomin [OLeary., 1980], $P_{k+1} = R_k - P_k(P_k^t A R_k)$.

Classical CG *vs.* Enlarged CG derived from Block CG

Algorithm 5 Classic CG

1: $r_0 = b - Ax_0$
2: $p_1 = \frac{r_0}{\sqrt{r_0^t A r_0}}$
3: **while** $||r_{k-1}||_2 > \varepsilon ||b||_2$ **do**
4: $\quad \alpha_k = p_k^t r_{k-1}$
5: $\quad x_k = x_{k-1} + p_k \alpha_k$
6: $\quad r_k = r_{k-1} - A p_k \alpha_k$
7: $\quad p_{k+1} = r_k - p_k (p_k^t A r_k)$
8: $\quad p_{k+1} = \frac{p_{k+1}}{\sqrt{p_{k+1}^t A p_{k+1}}}$
9: **end while**

messages per iteration
O(1) from SpMV +
O(log P) from dot prod + norm

Algorithm 6 ECG(Odir)

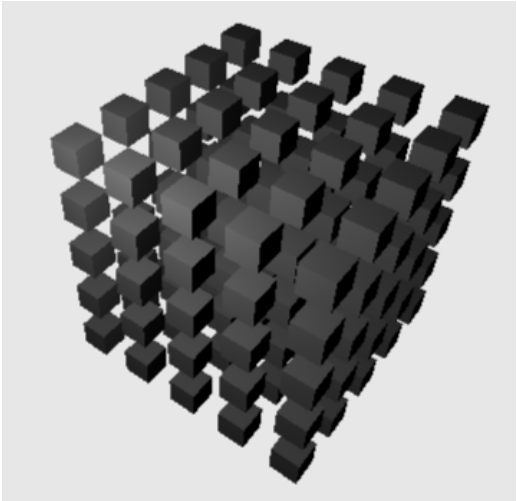
1: $R_0 = T(b - Ax_0)$
2: $P_1 = \text{A-orthonormalize}(R_0)$
3: **while** $|| \sum_{i=1}^{t} R_k^{(i)} ||_2 < \varepsilon ||b||_2$ **do**
4: $\quad \alpha_k = P_k^t R_{k-1}$ $\quad \triangleright\ t \times t$
5: $\quad X_k = X_{k-1} + P_k \alpha_k$ $\quad \triangleright\ n \times t$
6: $\quad R_k = R_{k-1} - A P_k \alpha_k$ $\quad \triangleright\ n \times t$
7: $\quad P_{k+1} = AP_k - P_k(P_k^t AAP_k) - P_{k-1}(P_{k-1}^t AAP_k)$ $\quad \triangleright\ n \times t$
8: $\quad P_{k+1} = \text{A-orthonormalize}(P_{k+1})$
9: **end while**
10: $x = \sum_{i=1}^{t} X_k^{(i)}$ $\quad \triangleright\ n \times 1$

messages per iteration
O(1) from SpMV +
O(log P) from BCGS + A-ortho

Test cases: boundary value problem

3D Skyscraper Problem - SKY3D

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



discretized on a 3D grid , where

$$\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1), if\ [10 * x_i] = 0 mod(2),\ i = 1, 2, 3, \\ 1, \qquad\qquad\qquad\qquad\qquad\qquad otherwise. \end{cases}$$

3D Anisotropic layers - ANI3D

- Ω divided into 10 layers parallel to $z = 0$, of size 0.1
- in each layer, the coefficients are constants (κ_x equal to 1, 10^2 or 10^4, $\kappa_y = 10\kappa_x$, $\kappa_z = 1000\kappa_x$).

Test cases (contd)

Linear elasticity 3D problem

$$\begin{aligned} \operatorname{div}(\sigma(u)) + f &= 0 && \text{on } \Omega, \\ u &= u_D && \text{on } \partial\Omega_D, \\ \sigma(u) \cdot n &= g && \text{on } \partial\Omega_N, \end{aligned}$$

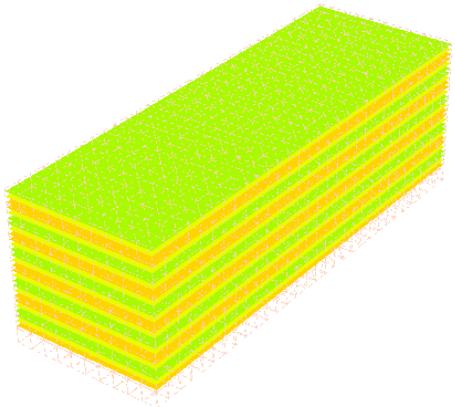


Figure : The distribution of Young's modulus

- $u \in \mathbb{R}^d$ is the unknown displacement field, f is some body force.
- Young's modulus E and Poisson's ratio ν take two values, $(E_1, \nu_1) = (2 \cdot 10^{11}, 0.25)$, and $(E_2, \nu_2) = (10^7, 0.45)$.
- Cauchy stress tensor $\sigma(u)$ is given by Hooke's law, defined by E and ν.

Test cases

Matrices

Generated with FreeFem++.

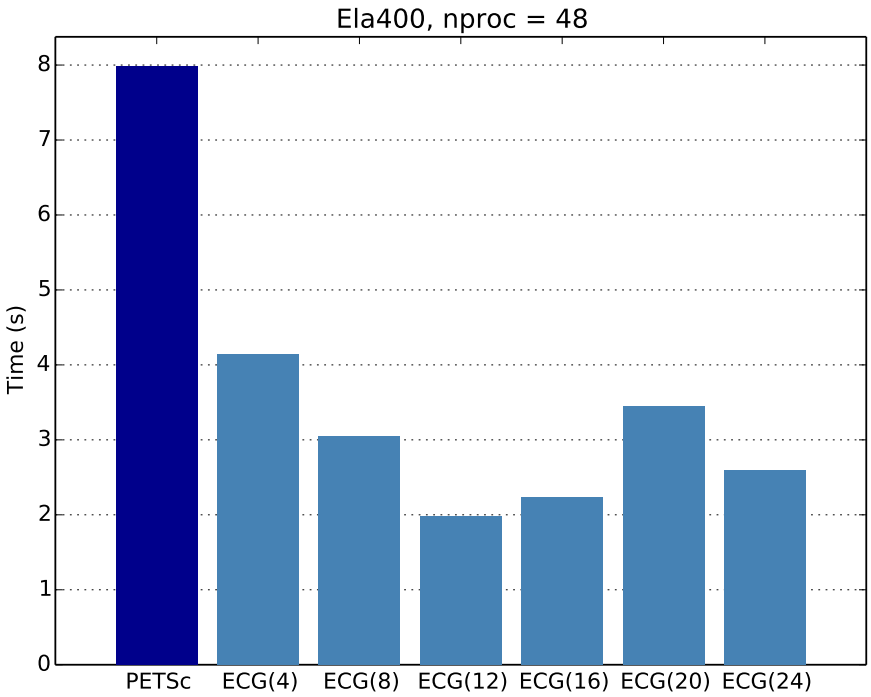
matrix	$n(A)$	$nnz(A)$	Description
SKY3D	8000	53600	Skyscraper
ANI3D	8000	53600	Anisotropic Layers
ELAST3D	11253	373647	Linear Elasticity P1 FE

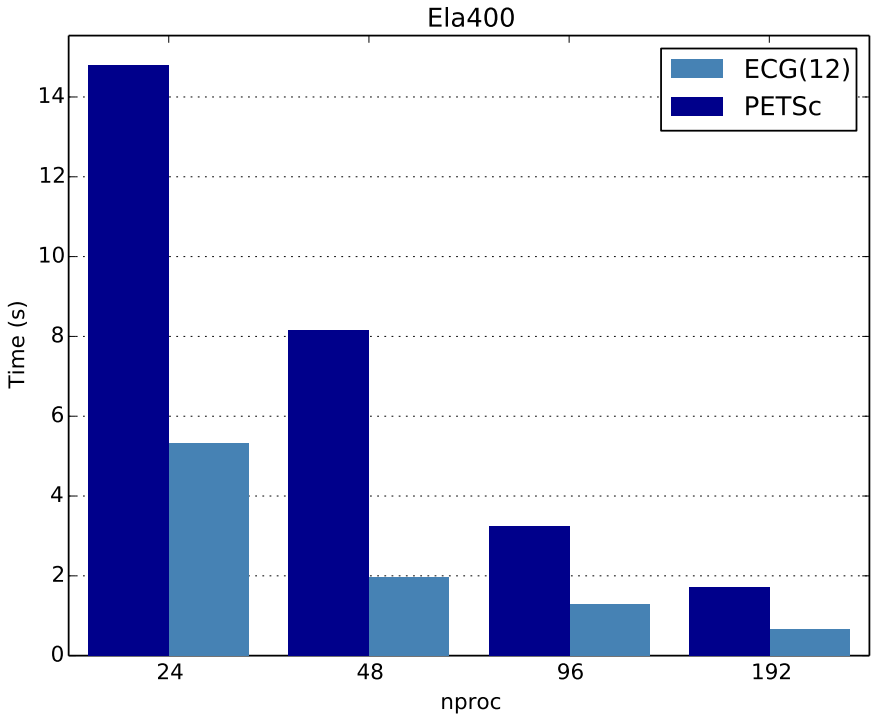
Convergence of different CG versions

Pa	CG Iter	CG Err	SRE-CG Iter	SRE-CG 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

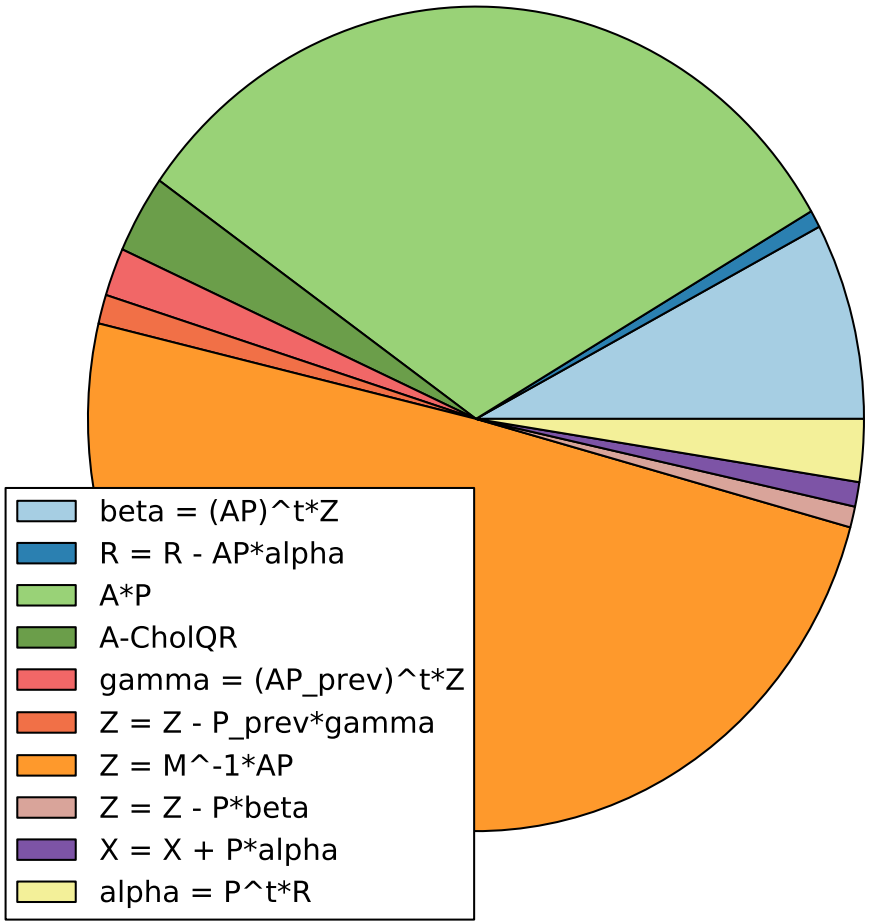
Comparison with PETSc

- Run on MeSU (UPMC cluster) $\rightarrow$ 24 cpus by node
- Compiled with Intel Suite 15, Petsc 3.7.4
- Results from [Grigori and Tissot, 2017]





Detailed profiling (source slide O. Tissot)



- Ela400 on 96 cores
- Orthodir ECG(12)
- Around 50% of the time spent in applying the preconditioner
- Around 30% of the time spent in Sparse Matrix-Matrix

Method	iter	time (s)	time/iter
ECG(12)	318	1.3	4.1×10^{-3}
PETSc	5198	3.3	6.3×10^{-4}

Table : Comparison with PETSc PCG. PETSc iteration is 6.5 times faster than ECG(12) one. MKL-Pardiso has a strange behaviour with multiple rhs n our experiments: 1 rhs solve is 3 times faster than 2 rhs solve.

References (1)

Ashby, S. F., Manteuffel, T. A., and Saylor, P. E. (1990).

A taxonomy for conjugate gradient methods.
SIAM Journal on Numerical Analysis, 27(6):1542–1568.

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. (2014a).

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

Grigori, L., Nataf, F., and Yousef, S. (2014b).

Robust algebraic Schur complement preconditioners based on low rank corrections.
Research Report RR-8557.

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.

Grigori, L. and Tissot, O. (2017).

Reducing the communication and computational costs of enlarged krylov subspaces conjugate gradient.
Research Report RR-9023.

References (2)

OLeary., D. P. (1980).

The block conjugate gradient algorithm and related methods.
Linear Algebra and Its Applications, 29:293–322.

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.