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

**Krylov subspace methods**

**Iterative solvers that reduce communication**
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, \( n = 105676, \text{nnz}(A) = 513072 \)

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 $x_1, x_2, \ldots, x_k$ that minimizes some measure of error over the corresponding spaces

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

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,
- $\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$ 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 $\kappa(A) = |\lambda_{\text{max}}(A)|/|\lambda_{\text{min}}(A)|$. 
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,

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

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: \( r_0 = b - Ax_0, \quad \rho_0 = \|r_0\|^2_2, \quad p_1 = r_0, \quad k = 1 \)
2: \textbf{while} ( \( \sqrt{\rho_k} > \varepsilon \|b\|_2 \) \text{ 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: \rho_k = \|r_k\|^2_2
11: 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

\[
\mathcal{K}_{k+1}(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^kr_0\},
\]

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
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

$$\mathcal{K}_{k+1}(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, ..., A^kr_0\},$$

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 $[x_1, ..., x_k]$

- Finds best solution $x_{k+1}$ as linear combination of $[x_1, ..., x_k]$

Typically, each iteration requires

- Sparse matrix vector product → point-to-point communication

- Dot products for orthogonalization → global communication
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 literature, but not presented here.
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. Kamgni (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)
GMRES: find $\mathbf{x}$ in $\text{span}\{\mathbf{b}, A\mathbf{b}, \ldots, A^k\mathbf{b}\}$ minimizing $||A\mathbf{x} - \mathbf{b}||_2$

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

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

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

\[
\begin{align*}
w &= A \cdot v(i-1) \\
\text{MGS}(w, v(0), \ldots, v(i-1)) \\
\text{update } v(i), H
\end{align*}
\]

endfor

solve LSQ problem with \( H \)

**Communication-avoiding GMRES**

\[
W = [ v, Av, A^2 v, \ldots , A^k v ]
\]

\([Q, R] = \text{TSQR}(W) \ldots \text{“Tall Skinny QR”}\)

Build \( H \) from \( R \), solve LSQ problem

**Sequential: #words\_moved =**

\[
O(k \cdot \text{nnz}) \text{ from SpMV} \\
+ O(k^2 \cdot n) \text{ from MGS}
\]

**Parallel: #messages =**

\[
O(k) \text{ from SpMV} \\
+ O(k^2 \cdot \log p) \text{ from MGS}
\]

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

\[
Ax = \begin{pmatrix}
  * & * & * \\
  * & * & * & * \\
  * & * & * & * & * \\
  * & * & * & * & * & * \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{pmatrix}
\begin{pmatrix}
  * \\
  * \\
  * \\
  * \\
  \vdots \\
\end{pmatrix} = \begin{pmatrix}
  * \\
  * \\
  * \\
  * \\
  \vdots \\
\end{pmatrix}
\]
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, \( n = 32, k = 3 \)
- Shaded triangles represent data computed redundantly

\[
Ax = \begin{pmatrix}
* & * & & & & \\
* & * & * & & & \\
* & * & * & * & & \\
& & * & * & * & \\
& & & & \cdots & \\
& & & & & \cdots
\end{pmatrix}
\begin{pmatrix}
* \\
* \\
* \\
* \\
\vdots \\
\vdots
\end{pmatrix} = \begin{pmatrix}
* \\
* \\
* \\
* \\
\vdots \\
\vdots
\end{pmatrix}
\]
Matrix Powers Kernel

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

\[
A x = \begin{pmatrix}
\ast & \ast & & & \\
\ast & \ast & \ast & & \\
& \ast & \ast & \ast & \\
& & \ddots & \ddots & \ddots \\
& & & \ast & \ast & \ast
\end{pmatrix}
\begin{pmatrix}
\ast \\
\ast \\
\ast \\
\vdots \\
\ast
\end{pmatrix}
= \begin{pmatrix}
\ast \\
\ast \\
\ast \\
\vdots \\
\ast
\end{pmatrix}
\]
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

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

<table>
<thead>
<tr>
<th>Proc 1</th>
<th>Proc 2</th>
<th>Proc 3</th>
<th>Proc 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A^3 \cdot x</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>A^2 \cdot x</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>A \cdot x</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>x</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
</tbody>
</table>

1 2 3 4 ... \ldots 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, \( n = 32, k = 3 \)
- Shaded triangles represent data computed redundantly

\[
Ax = \begin{pmatrix}
  * & * \\
  * & * & * \\
  * & * & * & * \\
  \vdots & \ddots & \ddots & \ddots \\
  \end{pmatrix}
\begin{pmatrix}
  * \\
  * \\
  * \\
  \vdots \\
  \end{pmatrix} = \begin{pmatrix}
  * \\
  * \\
  * \\
  \vdots \\
  \end{pmatrix}
\]
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, \( n = 32, k = 3 \)
- Shaded triangles represent data computed redundantly

\[
Ax = \begin{pmatrix}
* & * \\
* & * & * \\
* & * & * & *
\end{pmatrix}
\begin{pmatrix}
* \\
* \\
\vdots
\end{pmatrix} =
\begin{pmatrix}
* \\
* \\
\vdots
\end{pmatrix}
\]
Matrix Powers Kernel (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)

![Graph showing convergence of different polynomial bases](chart.png)

A different polynomial basis does converge: $[p_1(A)x, \ldots, p_k(A)x]$
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) with floor(restart length / k) == t.

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

Runtime values:
- k=5: 2.3x, 2.1x, 2.1x, 2.1x, 4.3x, 1.7x, 1.6x
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,

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

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

$$\mathcal{K}_{t,k}(A, r_0) = \text{span}\{T_s(r_0), AT_s(r_0), A^2 T_s(r0), \ldots, A^{k-1} T_s(r_0)\}$$

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

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

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

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

$$\mathcal{K}_{t,1}(A, r_0) \subset \ldots \subset \mathcal{K}_{t,k_{\text{max}}-1}(A, r_0) \subset \mathcal{K}_{t,k_{\text{max}}}(A, r_0) = \mathcal{K}_{t,k_{\text{max}}+q}(A, r_0), \forall q > 0.$$
Properties of enlarged Krylov subspaces: stagnation

- Let $K_{p_{\text{max}}} = K_{p_{\text{max}}} + q$ and $K_{t,k_{\text{max}}} = K_{t,k_{\text{max}}} + q$ for $q > 0$. Then $k_{\text{max}} \leq p_{\text{max}}$.

- The solution of the system $Ax = b$ belongs to the subspace $x_0 + K_{t,k_{\text{max}}}$.
Enlarged Krylov subspace methods based on CG

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

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

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

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

Result

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

\[ ||e_k||_A = ||x^* - x_k||_A \leq ||\bar{e}_k||_A \]
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.
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:

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

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

Total Flops $\approx 2\text{nnz} \cdot \bar{k}/t + 4n\bar{k}/t$

# words $\approx \mathcal{O}(\bar{k})$ (from SpMV)

# messages $\approx 2\, k \log(t) + \mathcal{O}(k)$ (from SpMV)

Cost of $k$ iterations of SRE-CG is:

Total Flops $\approx 2\text{nnz} \cdot k + \mathcal{O}(ntk)$

# words $\approx kt^2 \log(t) + \mathcal{O}(k)$ (from SpMV)

# messages $\approx k\log(t) + \mathcal{O}(k)$ (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.
Related work

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

\[ AX = B, \]

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

\[ \mathcal{K}_k(A, R_0) = \text{block} - \text{span}\{ R_0, AR_0, A^2 R_0, \ldots, A^{k-1} R_0 \}. \]

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

\[ AX = b \ast \text{ones}(1, t) \]

where \( X_0 \) is a block-vector containing the \( t \) initial guesses.
Algorithm 3 Classic CG

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

Algorithm 4 ECG(Odir)

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

- EK-CG based on Orthodir (Lanczos formula) [Ashby et al., 1990]
- More stable than Orthomin [OLeary., 1980], \( P_{k+1} = R_k - P_k (P_k^t A R_k) \).
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: \( \alpha_k = p_k^t r_{k-1} \)
5: \( x_k = x_{k-1} + p_k \alpha_k \)
6: \( r_k = r_{k-1} - A p_k \alpha_k \)
7: \( p_{k+1} = r_k - p_k (p_k^t A r_k) \)
8: \( p_{k+1} = \frac{p_{k+1}}{\sqrt{p_{k+1}^t A p_{k+1}}} \)
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: \( \alpha_k = P_k^t R_{k-1} \)
5: \( X_k = X_{k-1} + P_k \alpha_k \)
6: \( R_k = R_{k-1} - A P_k \alpha_k \)
7: \( P_{k+1} = A P_k - P_k (P_k^t A A P_k) - P_{k-1} (P_{k-1}^t A A P_k) \)
8: \( P_{k+1} = \text{A-orthonormalize}(P_{k+1}) \)
9: end while
10: \( x = \sum_{i=1}^t X_k^{(i)} \)

# messages per iteration
O(1) from SpMV +
O(log P) from BCGS + A-ortho
Test cases: boundary value problem

3D Skyscraper Problem - SKY3D

\[-\text{div}(\kappa(x)\nabla u) = f \text{ in } \Omega\]
\[u = 0 \text{ on } \partial\Omega_D\]
\[\frac{\partial u}{\partial n} = 0 \text{ on } \partial\Omega_N\]

discretized on a 3D grid, where

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

3D Anisotropic layers - ANI3D

- \(\Omega\) divided into 10 layers parallel to \(z = 0\), of size 0.1
- in each layer, the coefficients are constants (\(\kappa_x\) 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

\[
\text{div}(\sigma(u)) + f = 0 \quad \text{on } \Omega,
\]
\[
u = u_D \quad \text{on } \partial \Omega_D,
\]
\[
\sigma(u) \cdot n = g \quad \text{on } \partial \Omega_N,
\]

- \( u \in \mathbb{R}^d \) is the unknown displacement field, \( f \) is some body force.
- Young’s modulus \( E \) and Poisson’s ratio \( \nu \) 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 \( \nu \).
## Test cases

### Matrices

Generated with FreeFem++.

<table>
<thead>
<tr>
<th>matrix</th>
<th>$n(A)$</th>
<th>$nnz(A)$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKY3D</td>
<td>8000</td>
<td>53600</td>
<td>Skyscraper</td>
</tr>
<tr>
<td>ANI3D</td>
<td>8000</td>
<td>53600</td>
<td>Anisotropic Layers</td>
</tr>
<tr>
<td>ELAST3D</td>
<td>11253</td>
<td>373647</td>
<td>Linear Elasticity P1 FE</td>
</tr>
</tbody>
</table>
## Convergence of different CG versions

<table>
<thead>
<tr>
<th>Pa</th>
<th>SKY3D</th>
<th></th>
<th></th>
<th>ANI3D</th>
<th></th>
<th>ELAST3D</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CG</td>
<td></td>
<td></td>
<td>SRE-CG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pa</td>
<td>Iter</td>
<td>Err</td>
<td>Iter</td>
<td>Err</td>
<td>Iter</td>
<td>Err</td>
<td>Err</td>
</tr>
<tr>
<td>8</td>
<td>902</td>
<td>1E-5</td>
<td></td>
<td>211</td>
<td>1E-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>902</td>
<td>1E-5</td>
<td></td>
<td>119</td>
<td>9E-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>902</td>
<td>1E-5</td>
<td></td>
<td>43</td>
<td>4E-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4187</td>
<td>4e-5</td>
<td></td>
<td>875</td>
<td>7e-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4146</td>
<td>4e-5</td>
<td></td>
<td>673</td>
<td>8e-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4146</td>
<td>4e-5</td>
<td></td>
<td>449</td>
<td>1e-4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>4146</td>
<td>4e-5</td>
<td></td>
<td>253</td>
<td>2e-4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4146</td>
<td>4e-5</td>
<td></td>
<td>148</td>
<td>2e-4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>4146</td>
<td>4e-5</td>
<td></td>
<td>92</td>
<td>1e-4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1098</td>
<td>1e-7</td>
<td></td>
<td>652</td>
<td>1e-7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1098</td>
<td>1e-7</td>
<td></td>
<td>445</td>
<td>1e-7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1098</td>
<td>1e-7</td>
<td></td>
<td>321</td>
<td>8e-8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1098</td>
<td>1e-7</td>
<td></td>
<td>238</td>
<td>4e-8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1098</td>
<td>1e-7</td>
<td></td>
<td>168</td>
<td>5e-8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1098</td>
<td>1e-7</td>
<td></td>
<td>116</td>
<td>1e-8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison with PETSc

- Run on MeSU (UPMC cluster) → 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

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: 1 rhs solve is 3 times faster than 2 rhs solve.
A taxonomy for conjugate gradient methods.

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.

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

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

Reducing the communication and computational costs of enlarged krylov subspaces conjugate gradient.
Research Report RR-9023.
The block conjugate gradient algorithm and related methods.
*Linear Algebra and Its Applications, 29*:293–322.

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.