Row Projection Algorithms

Wei Deng, Nicole Eikmeier,
Nate Veldt, Xiaokai Yuan

Purdue University

April 28, 2016

Problem Setup

We seek solutions to:

$$\mathbf{Ax} = \mathbf{f}$$

- A is large, sparse, and may not be symmetric
- A can undergo a symmetric permutation to become banded

Reverse Cuthill-Mckee

We can perform a symmetric permutation with matrix P:

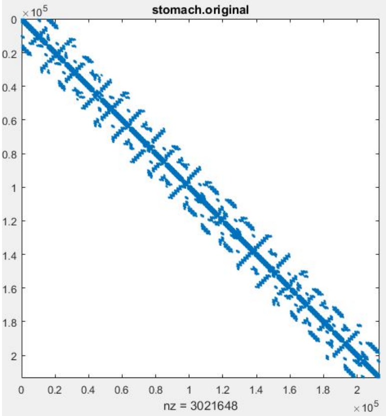


Figure : A sparse, non-symmetric

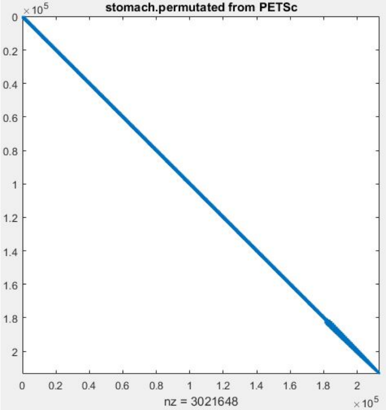


Figure : $P^T AP$ (banded)

If Bandwidth Too Large

For some problems the bandwidth could be too large, so instead we could choose P so that:

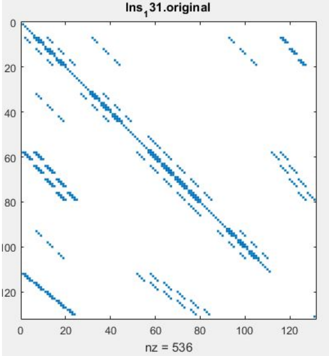


Figure : A is sparse and non-symmetric

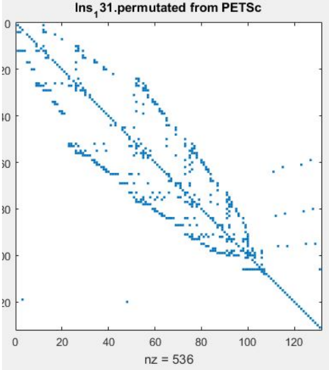


Figure : P^TAP narrow band+low rank

narrow band+low rank

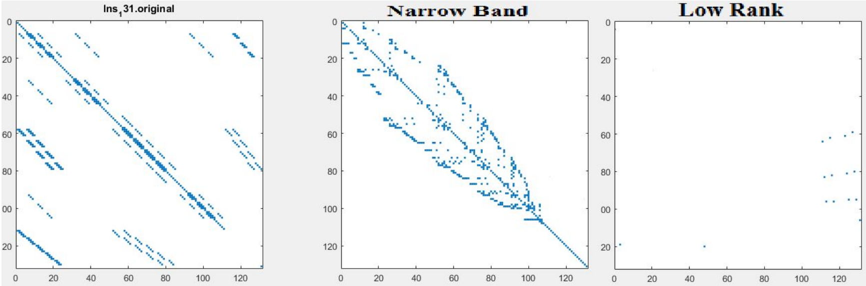


Figure : Breaking up A into a banded matrix plus a low rank matrix

Woodbury Formula

To solve

$$Ax = b \implies x = A^{-1}b$$

we use the following formula:

Woodbury Formula

$$\begin{aligned} A^{-1} &= (B - USV^T)^{-1} \\ &= B^{-1} - B^{-1}UTV^TB^{-1} \end{aligned}$$

where $T = (V^TB^{-1}U - S^{-1})^{-1}$

Solving $Ax = b$

Solving system:

$$
\begin{aligned}
x &= A^{-1}b \\
&= \mathbf{B^{-1}b} - B^{-1}UTV^T\mathbf{B^{-1}b} \\
&= \mathbf{a} - B^{-1}UT(V^T\mathbf{a}) \\
&= \mathbf{a} - B^{-1}UT\mathbf{c} \quad (\text{ solve } (V^TB^{-1}U - S^{-1})\mathbf{d} = \mathbf{c}) \\
&= \mathbf{a} - B^{-1}U\mathbf{d} \\
&= \mathbf{a} - B^{-1}\mathbf{h}
\end{aligned}
$$

All systems involving B are relatively easy to solve.

Alternatively, we could just get B and use it as a preconditioned for a Krylov subspace method.

Bandwidth after RCM reordering

Let's see the performance after the Reverse Cuthill Mckee

matrix	size	Matlab	PETSc	rcm.cpp
lns	131	32	× 111	× 113
ac2-db	21,982	545	×	×
bayer01	57,735	× 18,322	×	×
venkat25	62,424	1,515	1,515	1,495
stomach	213,360	1,133	2,216	2,239
atmosmodd	1,270,432	7,772	7,772	7,772

Re-ordering

To make the matrix better for parallelism, we do the following permutation

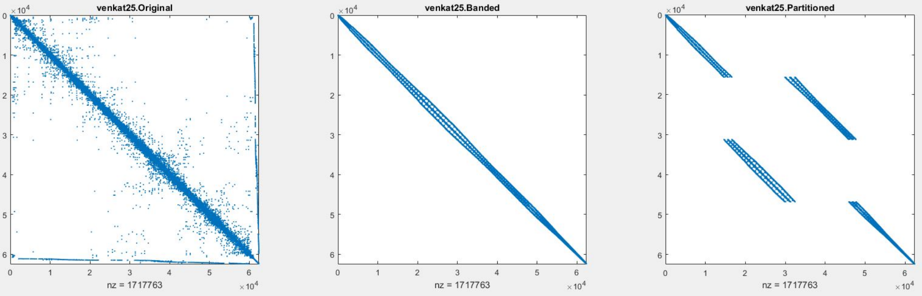
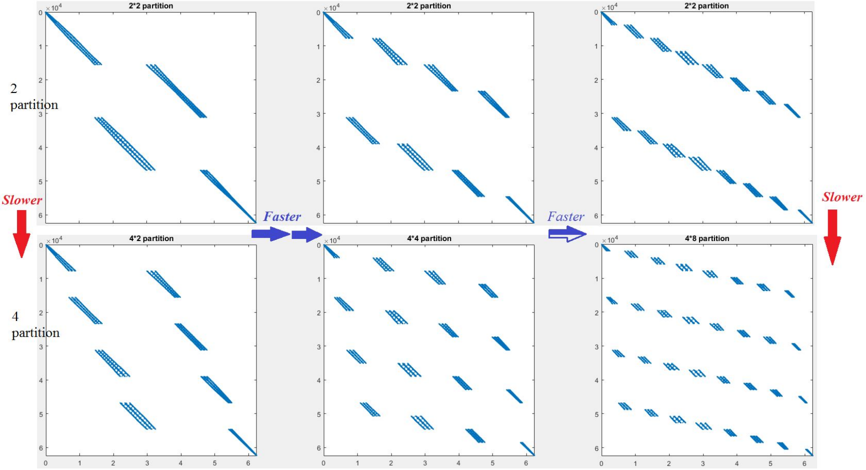


Figure : A is partitioned into 2 parts with several independent submatrices

Quality of Re-ordering

The test cases show that, 2 partion has the best performance, and with the number of blocks in each partition increasing, the performance goes better.



We can perform a symmetric permutation with matrix P:

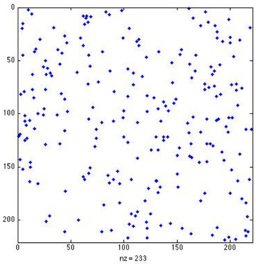


Figure : A sparse, non-symmetric

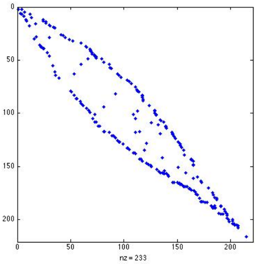


Figure : $P^T AP$ (banded)

If Bandwidth Too Large

For some problems the bandwidth could be too large, so instead we could choose P so that:

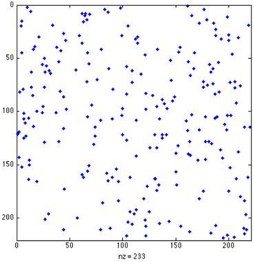


Figure : A is sparse and non-symmetric

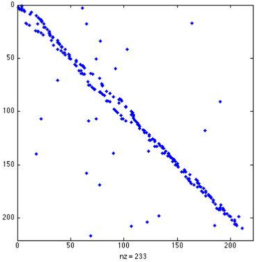
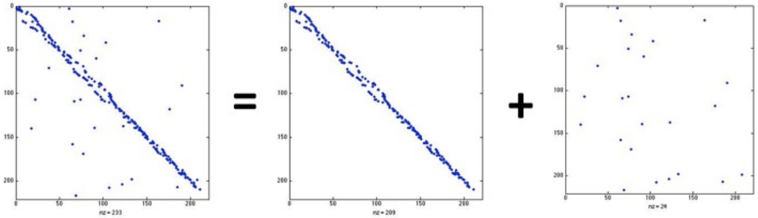


Figure : $P^T AP$ narrow band + low rank

Bandwidth

		Bandwidth	Bandwidth	Bandwidth
matrix	size	Matlab	PETSc	rcm.cpp
lns	131	32		113
ac2-db	21,982	545		×
bayer01	57,735			×
venkat25	62,424	1,515		1,495
stomach	213,360	1,133		2,239
atmosmodd	1,270,432	7,732		7,772



$$A = B + USV^T$$

Woodbury Formula

To solve

$$Ax = b \implies x = A^{-1}b$$

we use the following formula:

Woodbury Formula

$$\begin{aligned} A^{-1} &= (B - USV^T)^{-1} \\ &= B^{-1} - B^{-1}UTV^TB^{-1} \end{aligned}$$

where $T = (V^TB^{-1}U - S^{-1})^{-1}$

Solving $Ax = b$

Solving system:

$$
\begin{aligned}
x &= A^{-1}b \\
&= \mathbf{B^{-1}b} - B^{-1}UTV^T\mathbf{B^{-1}b} \\
&= \mathbf{a} - B^{-1}UT(V^T\mathbf{a}) \\
&= \mathbf{a} - B^{-1}UT\mathbf{c} \quad (\text{ solve } (V^TB^{-1}U - S^{-1})\mathbf{d} = \mathbf{c}) \\
&= \mathbf{a} - B^{-1}U\mathbf{d} \\
&= \mathbf{a} - B^{-1}\mathbf{h}
\end{aligned}
$$

All systems involving B are relatively easy to solve.

Alternatively, we could just get B and use it as a preconditioned for a Krylov subspace method.

Wei's Part

Kaczmarz method

Finding a common point of a set of hyperplanes $S_i = \{x : A_i x - b_i = 0\}$ for $i = 1, 2, ...,$ where A_i and b_i are ith row of matrix A and vector b

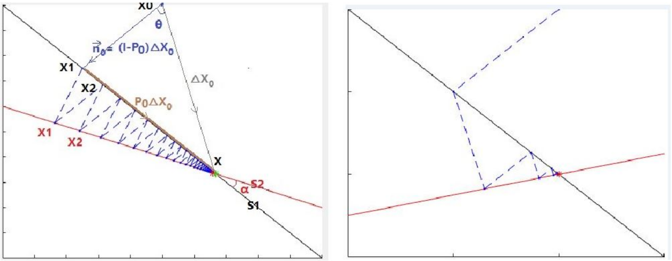


Figure : 2-Partition Case

So, the classical Kaczmarz method: $x_k = x_k + \overrightarrow{n_k} = x_k + \frac{r_k^i}{||A_i||^2} A_i^T$

where $\overrightarrow{n_k} = \triangle x_k cos\theta = \frac{\langle A_i, \triangle x_k \rangle}{||A_i||^2} A_i^T = \frac{b_i - \langle A_i, x_k \rangle}{||A_i||^2} A_i^T = \frac{r_k^i}{||A_i||^2} A_i^T$

Block Gauss-Seidel

$$\begin{cases} AA^T y = f \\ x = A^T y \end{cases} \Rightarrow AA^T = \begin{pmatrix} A_1^T \\ A_2^T \end{pmatrix} (A_1, A_2)$$

From Gauss-Seidel, use $x^{k+1} = (A_1, A_2) \begin{pmatrix} y_1^{k+1} \\ y_2^{k+1} \end{pmatrix}$

$$\begin{pmatrix} A_1^T A_1 & 0 \\ A_2^T A_1 & A_2^T A_2 \end{pmatrix} \begin{pmatrix} y_1^{k+1} \\ y_2^{k+1} \end{pmatrix} = \begin{pmatrix} 0 & -A_1 A_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_1^k \\ y_2^k \end{pmatrix} + \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

Replace some parameters with projection operator P_i, we can get:

$$x^{k+1} = A_1 y_1^{k+1} + A_2 y_2^{k+1} = Qx_k + b, \text{ where } Q = (I - P_2)(I - P_1)$$

Similarly, for symmetrized m-Partition:

$$x^{k+1} = Q_u x^k + f_u = (I - P_m)(I - P_{m-1})\ldots(I - P_1)x^k + f_u$$

However, the spectral radius of Q_u may interfere the convergence speed, and the distribution of eigenvalues could also influence the performance.

To handle this, we can symmetrize Q_u to get an accelerated iteration:

$$x^{k+1} = Q(\omega)x^k + Tf$$

where $Q(\omega) = (I - \omega P_1)(I - \omega P_2)\dots(I - \omega P_m)\dots(I - \omega P_2)(I - \omega P_1)$

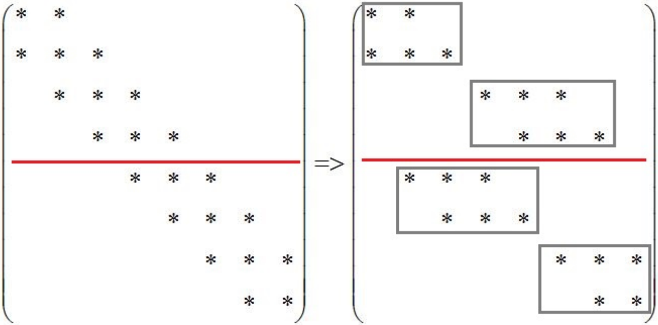
Since $I - Q(\omega)$ is S.P.D., the Conjugate Gradient method is suitable to accelerate the basic scheme.

Remark: it is also proved that the optimal value of ω is 1.0 and $Q(1)$ has the minimal spectral.

So the problem can be simplified as follows:

$$(I - Q(1))x = Tf$$

Permutation



New System After Permutation

Original non-symmetric system

$$\mathbf{Ax} = \mathbf{f}$$

A is large, sparse, and non-symmetric

New Symmetric Positive Definite System

$$(\mathbf{I} - \mathbf{Q})\mathbf{x} = \mathbf{c}$$

where

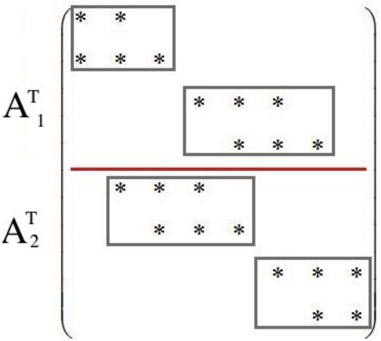
$$Q = (I - P_1)(I - P_2)\cdots(I - P_m)\cdots(I - P_1),$$

$$P_i = A_i(A_i^T A_i)^{-1}A_i^T$$

$$c = A^T(D + L)^{-T}D(D + L)^{-1}f$$

Computing c = Tf

$$A = \begin{bmatrix} A_1^T \\ A_2^T \end{bmatrix}, \; f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$



Computing c = Tf

$$\mathbf{Tf} = \Big[(I + (I - P_2)(I - P_1))(A_1^T)^+ \qquad (I - P_1)(A_2^T)^+\Big] \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = c$$

1. Solve pseudo-inverse problem:

$$v_i = (A_i^T)^+ f_i$$

2. Perform least squares computations of the form:

$$(I - P_i)x = y$$

Conjugate Gradient Method (Kamath and Sameh, 1988)

Step 1 : $x_0 = c$
Compute $r_0 = Tf - (I - Q)c = Qc$
Set $p_0 = r_0, i = 0$

Step 2: Compute:
$\alpha_i = (r_i, r_i)/(p_i, (I - Q)p_i)$
$x_{i+1} = x_i + \alpha_i p_i$
$\beta_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$
$p_{i+1} = r_{i+1} + \beta_i p_i$

Step 3: If convergence criterion is satisfied, terminate the iterations; else set $i = i + 1$ and return to Step 2.

We can take a convergence criterion as :

$$\frac{||r_i||}{||r_0||} \leq \epsilon$$

Key Step in the Framework is Least Squares Computations

Step 1 : $x_0 = c$

Compute $r_0 = Tf - \boxed{(I-Q)c} = Qc$

Set $p_0 = r_0, i = 0$

Step 2: Compute:

$$\alpha_i = (r_i, r_i)/(p_i, \boxed{(I-Q)p_i})$$
$$x_{i+1} = x_i + \alpha_i p_i$$
$$\beta_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$$
$$p_{i+1} = r_{i+1} + \beta_i p_i$$

Step 3: If convergence criterion is satisfied, terminate the iterations; else set $i = i + 1$ and return to Step 2.

We can take a convergence criterion as :

$$\frac{||r_i||}{||r_0||} \leq \epsilon$$

Compute $x_k = (I - P_i)x_k$

- It is not stable to form P_i directly since it will double the condition number of A_i, and at the same time will cost time on solving $(A_i^T A_i)^{-1}$.
- Given a vector u obtain $v = (I - P_j)u \Leftrightarrow \min_v ||u - A_j w||_2$.
- Solve $\min_v ||u - A_j w||_2$ directly: Normal equation is unstable, QR decomposition and SVD decomposition are too slow.
- Petsc: using CG method on normal equation: $A_j^T A_j w = A_j^T u$.
- Parallel step:

$$
\begin{aligned}
v &= \min_v ||u - A_j w||_2 \Leftrightarrow \min_{v_i} ||u_i - A_{j,i} w_i||_2, \\
v^T &= [v_1^T, v_2^T, ..., v_k^T], \\
w^T &= [w_1^T, w_2^T, ..., w_k^T].
\end{aligned}
$$

QR factorization for $Ax = b$

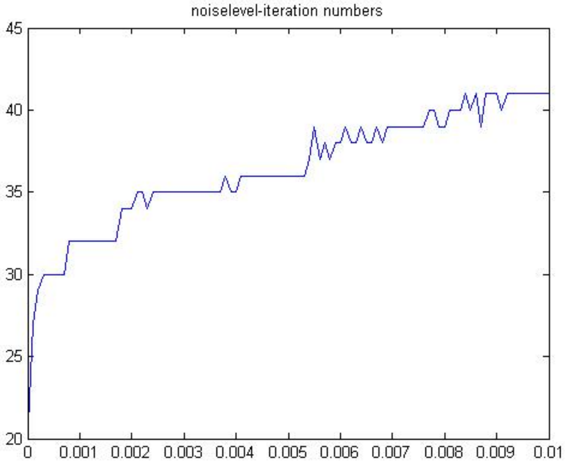
- $A_{m,n}$.
- $A = QR, \quad and \quad Rx = Q^T b.$
- QR factorization of A needs: $2mn^2 \quad flops$.
- form $d = Q^T b$ needs: $2mn \quad flops$.
- Solve $Rx = d$ by back substitution: $n^2 \quad flops$.
- for large m, n, the cost is about $2mn^2$.

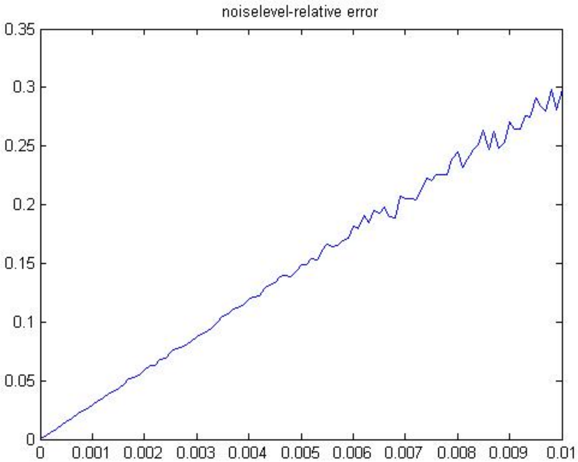
Cholesky factorization for $A^T Ax = A^T b$

- calculate $C = A^T A : 2n(n+1)(2m-1) \approx mn^2$ flops.
- Cholesky factorization $C = LL^T$: $\frac{1}{3}n^3$ flops.
- calculate $d = A^T b : 2mn$ flops.
- solve Lz = d by forward substitution :n^2 flops.
- solve $L^T x = z$ by back substitution :n^2 flops.
- cost for large m, n : $mn^2 + \frac{1}{3}n^3$ flops

sensitive of A p_k

- $x = ones(m, 1), A_{m,m}, eig(A) \in (0, 1), m = 1000, tolerance = 10^{-8}$.

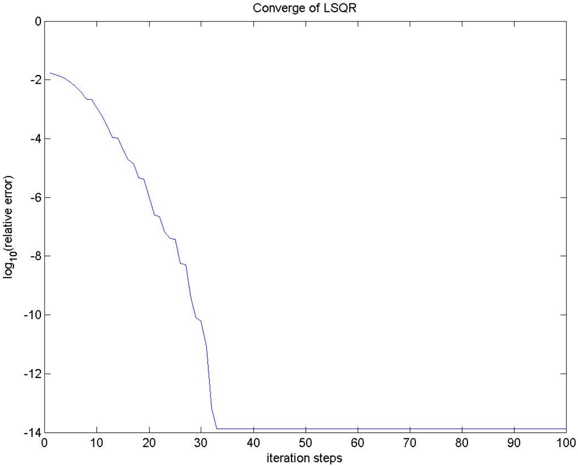



noiselevel-relative error
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01

LSQR and Lanczos process

- LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares, Christopher C.Paige, Michael A. Saunders.
- Step1, $\beta_1 u_1 = b, \alpha_1 v_1 = A^T u_1, w_1 = v_1, x_0 = 0, \bar{\phi_1} = \beta_1, \bar{\rho_1} = \alpha_1$. needs $2mn$ flop.
- Step2, bidiagonalization needs $4mn$ flop.
- $a.\beta_{i+1} u_{i+1} = Av_i - \alpha_i u_i$.
- $b.\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$.

- Step3, orthogonal transformation
- $a.\rho_i = \sqrt{(\bar{\rho_i^2} + beta_{i+1}^2)}$,
- $b.c_i = \bar{\rho}_i / \rho_i$,
- $c.s_i = \beta_{i+1} / \rho_i$,
- $d.\theta_{i+1} = s_i \alpha_{i+1}$,
- $e.\bar{\rho_{i+1}} = -c_i \alpha_{i+1}$,
- $f.\phi_i = c_i \bar{\phi}_i$,
- $g.\bar{\phi_{i+1}} = s_i \bar{\phi}_i$.
- Step4, Update x, w need $4n$ flop.
- $a.x_i = x_{i-1} + (\phi_i / \rho_i) w_i$,
- $b.w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i$.
- in total needs $6nm$ flop in each step.


Converge of LSQR
log10(relative error)
iteration steps
0
-2
-4
-6
-8
-10
-12
-14
0
10
20
30
40
50
60
70
80
90
100

Bandsize after Reverse-Cuthill McKee

matrix	size	band size
lns	131	32
std1-Jac2-db	21,982	545
bayer01	57,735	18,322
venkat25	62,424	1,515
stomach	213,360	1,133
atmosmodd	1,270,432	7,772

Bottle-neck - possibly improved by using the Woodbury Formula

Comparison to ILU Preconditioner

We re-ordered the matrix first using the MC64 software, then called a Krylov Subspace method with ILU pre-conditioner.

matrix	size	Converged?
lns	131	×
std1-Jac2-db	21,982	×
bayer01	57,735	×
venkat25	62,424	✓
stomach	213,360	✓
atmosmodd	1,270,432	✓

tol $= 10^{-8}$

Results for our Krylov Solver is using 4 MPI Processors.

		Our Parallel	Solver	Krylov	ILU
matrix	size	num-its	time (s)	num-its	time (s)
lns	131	10	.1	×	×
Jac2-db	21,982	13	.8	×	×
bayer01	57,735	×	×	×	×
venkat25	62,424	12	1.5	374	14.17
stomach	213,360	15	4	16	2.21
atmosmodd	1,270,432	14	21	266	130.72

Runtimes for Smaller Test Cases

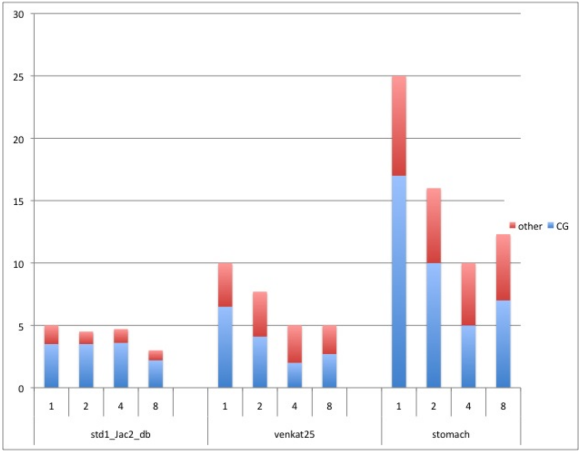


Figure : std1_Jac2bd: $n = 21982$; venkat25: $n = 62424$, stomach: $n = 213360$.

Runtimes for largest Test Case

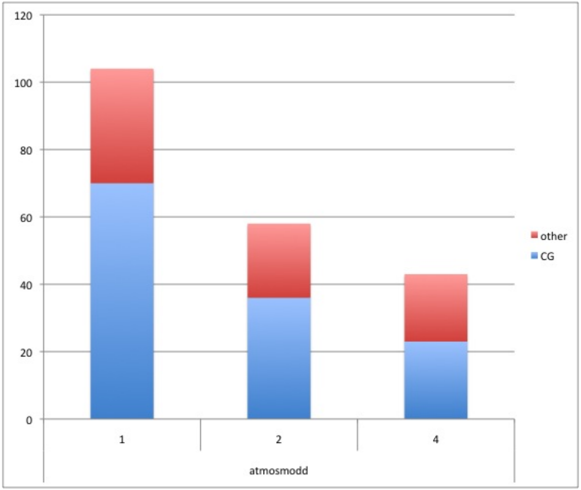


Figure : Runtime in seconds for largest test case ($n = 1270432$).

References

[1] Efstratios Gallopoulos, Bernard Philippe, and Ahmed H. Sameh. Pararallelism in Matrix Computations. Springer, 2016.

[2] Chandrika Kamath and Ahmed Sameh. A projection method for solving nonsymmetric linear systems on multiprocessors. Parallel Computing, 9:291-312, 1988.

[3] HSL, a collection of Fortran codes for large-scale scientific computation. See http://www.hsl.rl.ac.uk/