Group 4: The Kaczmarz Algorithm

Wei Deng, Nicole Eikmeier, Nate Veldt, and Xiaokai Yuan

May 2, 2016

1 Introduction

Project Objective:

Given a large sparse linear system $Ax = f$, of order $n = 10^6$ that can be effectively reduced to a banded matrix using the reordering scheme "reverse Cuthill-McKee", use the method of Row Projection, accelerated via the Conjugate Algorithm to yield a parallel solver. Compare the robustness and parallel scalability/speed with preconditioned Krylov subspace methods preconditioned via approximate LU-factorization.

The Kaczmarz Algorithm is a row projection method which is equivalent to solving $AA^Ty = f$ using the Gauss-Siedel iteration, with $x = A^Ty$. In Kaczmarz we consider each equation as a hyperplane:

$$S_i = \{x : A_i x - b_i = 0\}$$

for $i = 1, 2, ...$, where A_i and b_i are ith row of matrix A and vector b. So the problem is transformed into finding the coordinates of the point of intersection of these hyperplanes. See figure 1 for a visualization of the problem.

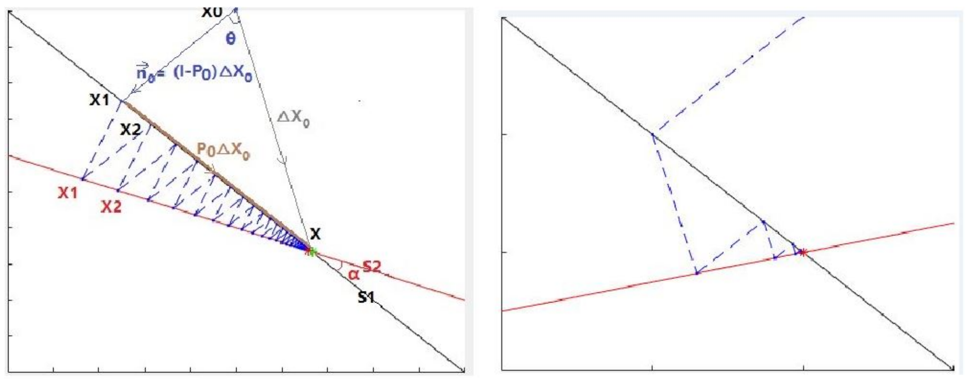

Figure 1: 2-Partition Case of the Kaczmarz Algorithm

From this basic idea we use symmetrization and acceleration via the CG method to improve the procedure. The details will be given carefully in the following sections.

2 Implementation

2.1 Reverse Cuthill-McKee and the Woodbury Formula

The first step in our implementation is to transform the sparse matrix A to a banded form using Reverse Cuthill-McKee. We can perform a symmetric permutation with matrix P so that $P^T AP$ is banded. In figure 3, we see the result from Matlab of Reverse Cuthill Mckee on the stomach data set. The band size is nice and small.

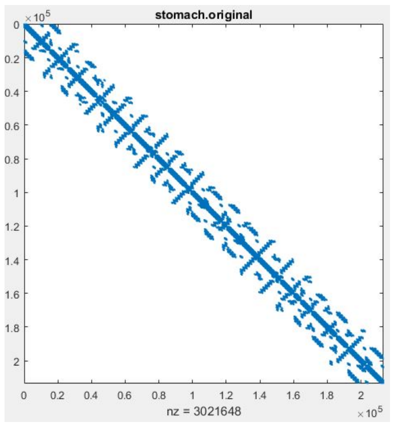

Figure 2: A sparse, non-symmetric

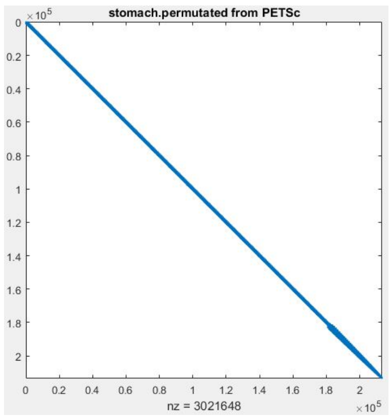

Figure 3: $P^T AP$ (banded)

For some problems the bandwidth could be too large after using Reverse Cuthill Mckee. See for example figure 5, where we see the result of RCM on the lns_131 data set, again in Matlab. If this is the case, instead we could choose P so that $P^T AP$ is narrow banded plus a low rank matrix. See figure 6.

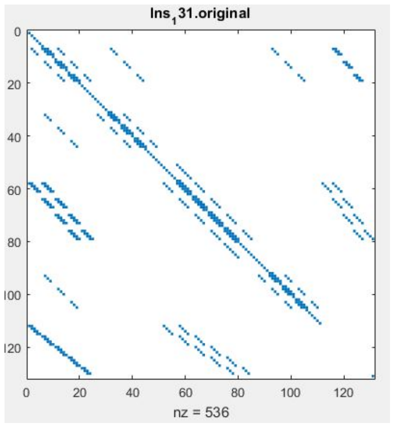

Figure 4: A is sparse and non-symmetric

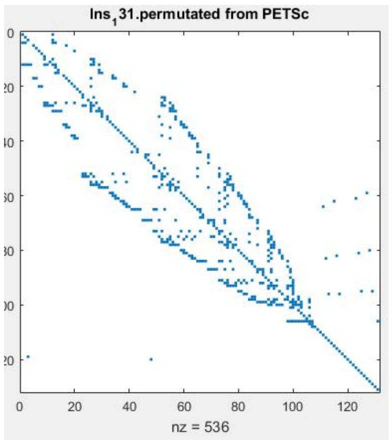

Figure 5: $P^T AP$ narrow band+low rank

In this case we use the Woodbury Formula to solve $Ax = b$. Recall first that $Ax = b$ implies $x = A^{-1}b$. Then,

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

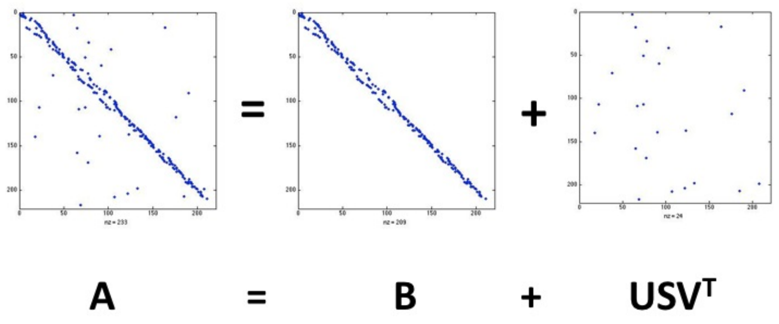

Figure 6: Breaking up A into a banded matrix plus a low rank matrix

matrix	size	Matlab	PETSc	rcm.cpp
lns	131	32	× 111	× 113
Jac2-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

Table 1: Bandsize after RCM using various codes

where $T = (V^T B^{-1} U - S^{-1})^{-1}$. And

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

The Woodbury formula is useful because all systems involving B are relatively easy to solve. We did not implement the Woodbury formula in our experiments, due to time contraints. This means that we transformed A to a completely banded matrix, no matter how large the band size. We expect there would be improvements in our final results if the Woodbury formula was implemented.

In table 1, the band size after permutation is shown. Matlab performed the best across the board, but all three permutations did well for our largest test case. You can see also that for the bayer01 matrix, our band size is very large, and so we did not get reasonable results in our parallel Kaczmarz implementation.

2.2 The Kaczmarz method

Once we have a banded matrix, A, we move on to the Kaczmarz algorithm, which was introduced earlier. Suppose we partition A into two pieces

$$A = \begin{pmatrix} A_1^T \\ A_2^T \end{pmatrix}$$

and we also partition the right hand size vector $f^T = (f_1^T, f_2^T)$.

In the classical Kaczmarz method at each iteration we compute:

$$x_k = x_k + \overrightarrow{n_k} = x_k + \frac{r_k^i}{||A_i||^2} A_i^T$$

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

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

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

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

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

$$x^{k+1} = A_1 y_1^{k+1} + A_2 y_2^{k+1} = Q x_k + b$$

where $Q = (I - P_2)(I - P_1)$. Similarly, for an 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 with the convergence speed, and the distribution of eigenvalues could also influence the performance. To handle this, we can symmetric Q_u to get an accelerated iteration:

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

where $Q(\omega) = (I - \omega P_1)(I - \omega P_2) \ldots (I - \omega P_m) \ldots (I - \omega P_2)(I - \omega P_1)$, and $T(\omega) = A^T (D + \omega L)^{-T} D (D + \omega L)^{-1}$, where $A^T A = L + D + L^T$ is the splitting into block lower, block upper, and block diagonal pieces of $A^T A$. Since $(I - Q)$ is symmetric positive definite, the conjugate gradient method is suitable to accelerate the basic scheme.

Once we have a banded matrix, we considered using various partitions of the rows. A very simple example is shown in figure 7. A permutation like this gives several benefits. The first is that we can have an outer level of parallelism for each projection. Also, when we split the matrix this way we create several small independent least squares problems, which reduces the time.

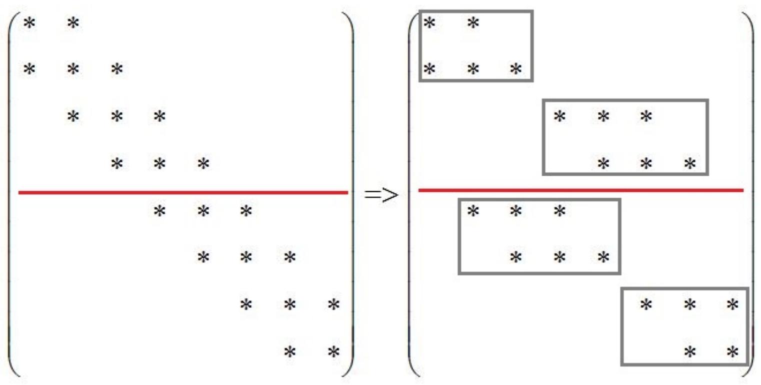

Figure 7: Re-ordering the banded matrix

In Figure 8 we consider different types of partitions, both in the inner and outer levels. We found that have more independent blocks resulted in faster implementation of Kaczmarz. We also found that more partitions, i.e. taking 4 partitions:

$$A = \begin{pmatrix} A_1^T \\ A_2^T \\ A_3^T \\ A_4^T \end{pmatrix}$$

instead of 2 partitions is slower.

Furthermore, it has been proven that the optimal value of ω is 1.0 if we use two partitions, and $Q(1)$ has the minimal spectral radius. For these reasons we decided to use 2 outer partitions for our implementation. In this case the problem can be simplified as follows:

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

2.3 Calculation of c

To transform the system $Ax = f$ into solving the symmetric positive definite system $(I - Q)x = c$, we need to get the new right hand side vector c. Recall that we use $m = 2$ partitions in our work. We let

$$A = \begin{pmatrix} A_1^T \\ A_2^T \end{pmatrix}$$

and we also partition the right hand size vector

$$f^T = (f_1^T, f_2^T).$$

For two partitions, if we expand $c = Tf$ to be:

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

We see that in order to perform this we need to solve two pseudo inverse problems:

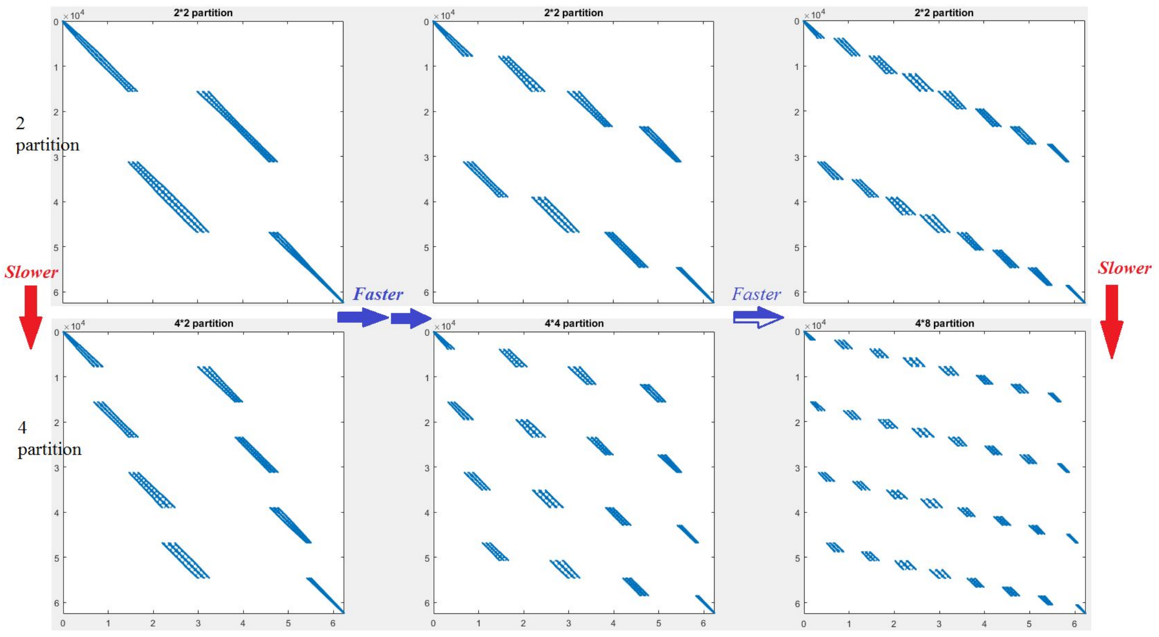

Figure 8: Speed for different partition types.

$$\hat{f}_i = (A_i^T)^+ f_i$$

and whenever we see a product of the form $(I - P_i)u = v$ we are really performing a least squares operation involving the block A_i. We will discuss this least squares problem in more detail later.

Computing c can be accomplished by solving a saddle point problem. In our implementation we spent some of our efforts trying to compute c, but due to time constraints and difficulties with other parts of our algorithm we were not able to ever correctly incorporate a proper computation of c in our work. For this reason, for our code we generated a right hand side vector c by setting u to be the all ones vector, and then performing $c := (I - Q) \cdot u$ via a sequence of least squares operations. Then we were able to use our conjugate gradient framework to solve the system $(I - Q)x = c$.

In general for Kaczmarz we expect the time taken in running the CG scheme and calling the least squares function to be the bottleneck in our computations. So even though we were unable to properly generate the right hand side vector from a starting vector f, we chose to instead focus on the CG scheme and least squares framework in our implementation.

2.4 Least Squares Computations

Recall that the new system we are dealing with is $(I - Q)x = c$, where $c = Tf$, and

$$Q = (I - P_1)(I - P_2)(I - P_1)$$

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

It is not stable to form P_i directly since it will square the condition number of A_i, and at the same time will cost time in solving a system with the matrix $(A_i^T A_i)$. Instead, given a vector u we note that $v = (I - P_j)u \Leftrightarrow \min_v ||u - A_j w||_2$. We therefore solve $\min_v ||u - A_j w||_2$ to obtain the vector v.

Because of the way we have split our matrix, when we solve a least squares problem $(I - P_i)u = v$ we in theory can do so independently because A_i is a matrix with many independent blocks. For proper parallel implementation, we would perform the following steps:

1. At the beginning of the algorithm, send different blocks of A_i (for each sub matrix i) to different processors.
2. In each least squares computation, independently perform least squares operations on the blocks, and then gather the final result after the simultaneous, independent least squares operations.

QR Factorizations

One very effective way to compute the least squares problem in parallel is to use the QR factorization via Given's rotations on each independent block of each sub matrix A_i. The computation and storage of the QR factorization of the entire matrix Q would be extremely expensive. However, because A_i is made up of independent diagonal blocks that are all very small in size, we can perform QR factorizations of each of these blocks with great success.

There are two clear benefits of using the QR factorization on the independent sub blocks of A_i. First of all, we can find this factorization simultaneously for each block of A_i at the very beginning of the algorithm, then in all future calls to our least squares function we can use the factorization without needing to recompute it. Secondly, the storage required for the factorization of the blocks is very small in comparison with the overall system.

Difficulties

Unfortunately in our project we were not able to make full use of the parallelism that is theoretically possible by the Kaczmarz framework. Our difficulty was that splitting up a parallel matrix A_i into independent pieces is extremely difficult in PETSc. This causes a number of issues in our final code, including the following:

- We cannot be sure that the blocks are perfectly split up among different processors, since PETSc assigns rows automatically by itself without our control.
- Even though we have multiple processors at work on a least squares problem, the least squares computation is still performed globally on the entire sub matrix A_i, rather than simultaneously on independent blocks of A_i.
- We are unable to factorize pieces of A_i ahead of time to save time in later calls to our sequential least squares algorithm

With a better knowledge of PETSc these problems could be remedied, but time constraints rendered it impossible to master the software enough so that we could reach the full potential of the Kaczmarz framework.

2.4.1 Attempt to Improve Speed by LSQR

In the end, in order to solve the least squares problem, we implemented LSQR. LSQR is an iterative method for solving $Ax = b$ where A is large, sparse, and overdetermined. It is equivalent at each iteration to CG.

Algorithm 1 LSQR

```
1: $\beta_1 u_1 = b, \alpha_1 v_1 = A^T u_1, w_1 = v_1, x_0 = 0, \bar{\phi}_1 = \beta_1, \bar{\rho}_1 = \alpha_1$
2: for $i = 1, 2, 3, \cdots$ do
3:     Bidiagonalization:
4:     $\beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i$
5:     $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$
6:     Orthogonal Transformation:
7:     $\rho_i = \sqrt{(\bar{\rho}_i^2 + \beta_{i+1}^2)}$
8:     $c_i = \bar{\rho}_i / \rho_i,$
9:     $s_i = \beta_{i+1} / \rho_i,$
10:    $\theta_{i+1} = s_i \alpha_{i+1},$
11:    $\bar{\rho}_{i+1} = -c_i \alpha_{i+1},$
12:    $\phi_i = c_i \bar{\phi}_i,$
13:    $\bar{\phi}_{i+1} = s_i \bar{\phi}_i.$
14:    Update x:
15:    $x_i = x_{i-1} + (\phi_i / \rho_i) w_i,$
16:    $w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i.$
```

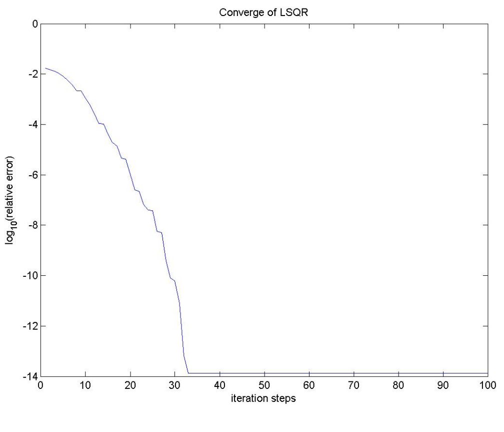

Figure 9: Convergence of LSQR

We had hoped that using this method and truncating the number of iterations used in the least squares calculations would allow us to obtain good overall results for our Kaczmarz algorithm. Note the plot of the convergence of the LSQR method in Figure 9. We can see that the algorithm mostly converges in the first several iterations, so we hoped that by truncating the number of steps taken we would make the least squares computation faster without harming the accuracy too much. We know that Krylov subspace methods are robust against poor matrix-vector products, so the hope was that our outer CG scheme would still progress towards an accurate overall solution of $(I-Q)x = c.$. However, our final results are still far from satisfactory in terms of accuracy if we truncate the computations too much.

One thing we tried was to increase the maximum number of inner LSQR iterations used as the number of outer CG scheme iterations increased. We did this to try to find a balance between speed and accuracy. However, to achieve any type reasonable accuracy for our larger test cases, we did eventually need to make the iterative least squares method run for many iterations (over 100 for each least squares computation). This leads to very slow runtimes. We provide our results in a later section.

2.5 Acceleration via the CG Method

As stated previously, since $(I - Q)$ is symmetric positive definite, we accelerate solving $(I - Q)x = Tf$ using the conjugate gradient method. This is the framework of our entire algorithm. The outline follows:

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 :

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

The boxed out portions in our framework are very important. Remember that whenever we see:

$$(I - Q)u$$

for any vector u, we really mean to solve multiple least squares problems:

$$(I - Q)u = (I - P_1)(I - P_2)(I - P_1)u$$

$$(I - P_i)u \Rightarrow \min_v ||u - A_i w||$$

3 Results and Conclusions

3.1 Results

We will now discuss the results of our parallel solver. Because of our slow least squares computations, we did not reach a very low tolerance for any of our test cases. For this

		Our Parallel	Solver	Krylov	ILU
matrix	size	tol achieved	time (s)	num-its	time (s)
lns	131	$7.5E{-}03$	5.6	×	×
Jac2-db	21,982	$2.0E{-}04$	283	×	×
venkat25	62,424	$2.5E{-}03$	874	374	14.17
stomach	213,360	$2.1E{-}06$	194	16	2.21
atmosmodd	1,270,432	$3.0E{-}03$	7100	266	130.72

Table 2: Runtimes of our algorithm (100 iterations of CG) in comparison with GMRES

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

Table 3: ILU preconditioner

reason we chose to simply truncate our CG loop at 100 iterations and report the runtime and the tolerance attained by this point. The results are given in table 2.

To find a benchmark of how our method performed, we compared with a Krylov Subspace method. The implementation of this consisted of first using the MC64 software to re-order the matrix. This software finds a permutation which brings non-zeros to the diagonal. Once this permutation is used, then we called a Krylov Subspace method with ILU pre-conditioner in PETSc. The Krylov subspace method used is GMRES.

Note that the ILU preconditioner fails for some of our data. The reason for this could be that the eigenvalues of these matrices are distributed too much across the positive and negative spectrum. The ILU preconditioner is not well equipped to handle these so called "highly indefinite" matrices. Table 3 shows which of our data failed with this method.

We note that the one success of our algorithm was its ability to make progress in solving the first few test cases where we failed to converge for GMRES. We see that the framework of our algorithm is good for solving these difficult systems, though runtimes are terrible for reasons already addressed.

3.2 Conclusions

In conclusion, it was a mistake to use PETSc. PETSc is better for high level implementation, not for optimization and efficiency. PETSc was also not well suited specifically for grabbing the exact sub-matrices that we needed in order to call the parallel Least Squares.

In the end we tried to salvage our project by implementing LSQR, but even with this attempt we did not obtain the desired outcome. If we had left PETSc out of our project we know our results would have been even better.

References

[1] Hsl, a collection of fortran codes for large-scale scientific computation.

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

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

[4] Christopher C. Paige and Michael A. Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, March 1982.