Parallel Singular Value Decomposition

Jiaxing Tan

Outline

- What is SVD?
- How to calculate SVD?
- How to parallelize SVD?
- Future Work

What is SVD?

The SVD is the Swiss Army knife of matrix decompositions

—Diane O'Leary, 2006

Matrix Decomposition

- Eigen Decomposition
- A (non-zero) vector **v** of dimension *N* is an **eigenvector** of a square (*N*×*N*) matrix **A** if it satisfies the linear equation

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

- Where λ is **eigenvalue** corresponding to **v**

- What if it is not a square matrix?

What is SVD?

- Singular Value Decomposition(SVD) is a technique for factoring matrices.

- Now comes a highlight of linear algebra. Any real m × n matrix can be factored as

$$A = U\Sigma V^T$$

- Where U is an m×m orthogonal matrix whose columns are the eigenvectors of AA^T , V is an n×n orthogonal matrix whose columns are the eigenvectors of A^TA

What is SVD?

- Σ is an m × n diagonal matrix of the form

$$\Sigma = \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & 0 & \\ & & \sigma_r & & \\ & & & 0 & \\ & 0 & & & \ddots \\ & & & & & 0 \end{pmatrix}$$

- with σ1 ≥ σ2 ≥ · · · ≥ σr > 0 and r = rank(A). In the above, σ1, . . . , σr are the square roots of the eigenvalues of AT A. They are called the singular values of A.

What is SVD?

- Factor matrix A of size MxN $A = U\Sigma V^T$
  - – U Is size M*M and Contains left Singular values
  - – Σ is size M*N and Contains singular Values of A
  - – V is size N*N and contains the right singular values

SVD and Eigen Decomposition

- Intuitively, SVD says for any linear map, there is an orthonormal frame in the domain such that it is first mapped to a different orthonormal frame in the image space, and then the values are scaled.
- Eigen decomposition says that there is a basis, it doesn't have to be orthonormal, such that when the matrix is applied, this basis is simply scaled. That is assuming you have n linearly independent eigenvectors of course.

Example of SVD

- Singular value decomposition takes a rectangular matrix of gene expression data (defined as A, where A is a *n* x *p* matrix) in which the *n* rows represents the genes, and the *p* columns represents the experimental conditions.
- The SVD theorem states:

$$\mathbf{A}_{nxp} = \mathbf{U}_{nxn}\,\mathbf{S}_{nxp}\,\mathbf{V}^{\mathbf{T}}_{pxp}$$

- Where the columns of U are the left singular vectors (*gene coefficient vectors*); S (the same dimensions as *A*) has singular values and is diagonal (*mode amplitudes*); and V^T has rows that are the right singular vectors (*expression level vectors*).

Example

Index Words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
book			1	1					
dads						1			1
dummies		1						1	
estate							1		1
guide	1					1			
investing	1	1	1	1	1	1	1	1	1
market	1		1						
real							1		1
rich						2			1
stock	1		1					1	
value				1	1				

book	0.15	-0.27	0.04
dads	0.24	0.38	-0.09
dummies	0.13	-0.17	0.07
estate	0.18	0.19	0.45
guide	0.22	0.09	-0.46
investing	0.74	-0.21	0.21
market	0.18	-0.30	-0.28
real	0.18	0.19	0.45
rich	0.36	0.59	-0.34
stock	0.25	-0.42	-0.28
value	0.12	-0.14	0.23

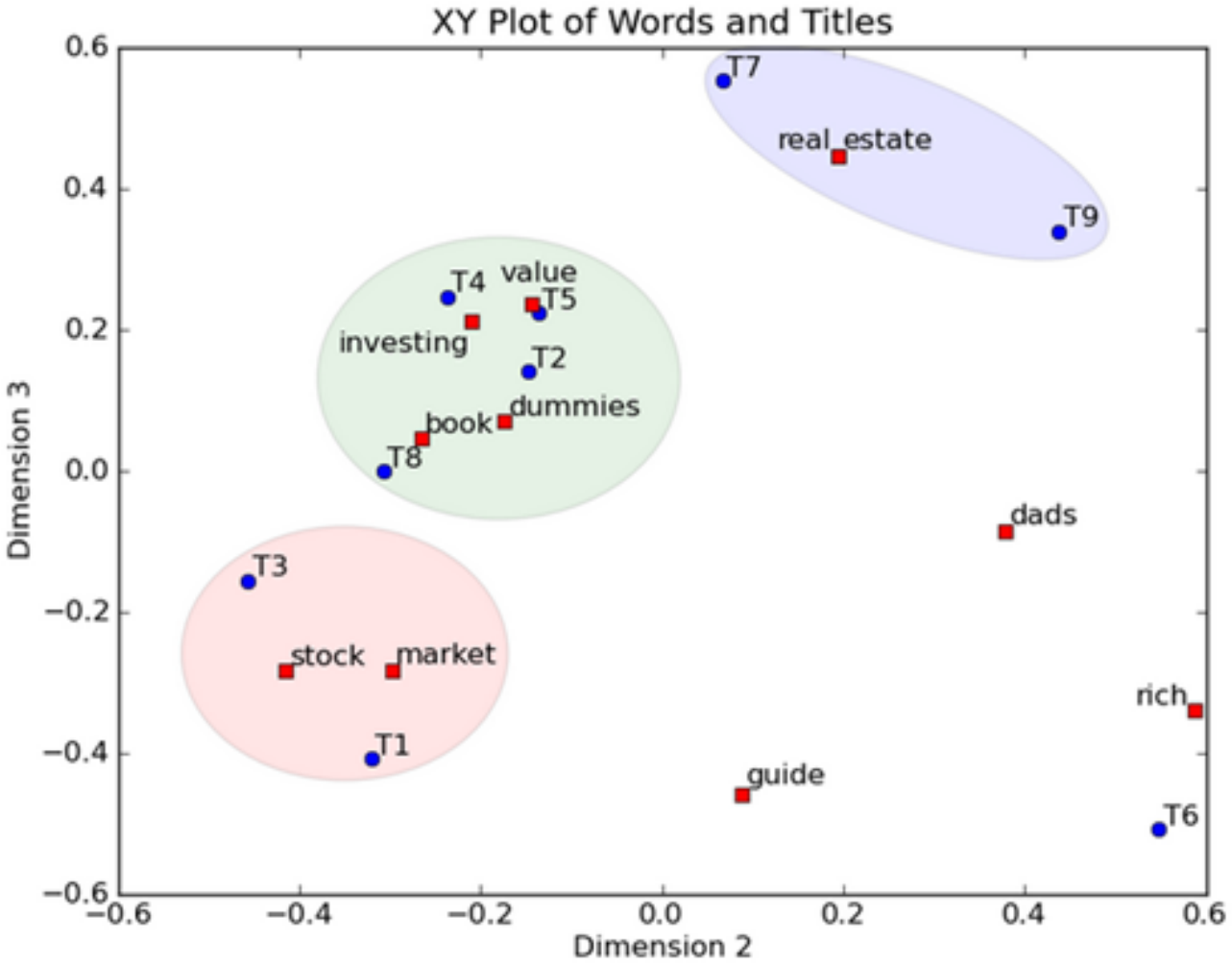
*

3.91	0	0
0	2.61	0
0	0	2.00

*

T1	T2	T3	T4	T5	T6	T7	T8	T9
0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0.00	0.34

Example


XY Plot of Words and Titles
Dimension 3
Dimension 2
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
T7
real_estate
T9
T4
value
T5
investing
T2
dummies
book
T8
dads
T3
stock
market
rich
T1
guide
T6

How to Calculate SVD

An **eigenvector** of a square matrix $\mathbf{A}$ is a vector $\mathbf{v}$ such that $\mathbf{A}$ only changes the magnitude of $\mathbf{v}$

- I.e. $\mathbf{Av} = \lambda\mathbf{v}$ for some $\lambda \in \mathbb{R}$
- Such λ is an **eigenvalue** of $\mathbf{A}$

The **eigendecomposition** of $\mathbf{A}$ is $\mathbf{A} = \mathbf{Q}\boldsymbol{\Delta}\mathbf{Q}^{-1}$

- The columns of $\mathbf{Q}$ are the eigenvectors of $\mathbf{A}$
- Matrix $\boldsymbol{\Delta}$ is a diagonal matrix with the eigenvalues

Not every (square) matrix has eigendecomposition

- If $\mathbf{A}$ is of form $\mathbf{BB}^T$, it always has eigendecomposition

The SVD of $\mathbf{A}$ is closely related to the eigendecompositions of $\mathbf{AA}^T$ and $\mathbf{A}^T\mathbf{A}$

- The left singular vectors are the eigenvectors of $\mathbf{AA}^T$
- The right singular vectors are the eigenvectors of $\mathbf{A}^T\mathbf{A}$
- The singular values are the square roots of the eigenvalues of both $\mathbf{AA}^T$ and $\mathbf{A}^T\mathbf{A}$

How to Calculate SVD

Example: Find the SVD of A, $U\Sigma V^T$, where $A = \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix}$.

First we compute the singular values σ_i by finding the eigenvalues of AA^T.

$$AA^T = \begin{pmatrix} 17 & 8 \\ 8 & 17 \end{pmatrix}.$$

The characteristic polynomial is $det(AA^T - \lambda I) = \lambda^2 - 34\lambda + 225 = (\lambda - 25)(\lambda - 9)$, so the singular values are $\sigma_1 = \sqrt{25} = 5$ and $\sigma_2 = \sqrt{9} = 3$.

How to Calculate SVD

Now we find the right singular vectors (the columns of V) by finding an orthonormal set of eigenvectors of A^TA. It is also possible to proceed by finding the left singular vectors (columns of U) instead. The eigenvalues of A^TA are 25, 9, and 0, and since A^TA is symmetric we know that the eigenvectors will be orthogonal.

For $\lambda = 25$, we have

$$A^TA - 25I = \begin{pmatrix} -12 & 12 & 2 \\ 12 & -12 & -2 \\ 2 & -2 & -17 \end{pmatrix}$$

which row-reduces to $\begin{pmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$. A unit-length vector in the kernel of that matrix is $v_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix}$.

How to Calculate SVD

For $\lambda = 9$ we have $A^TA - 9I = \begin{pmatrix} 4 & 12 & 2 \\ 12 & 4 & -2 \\ 2 & -2 & -1 \end{pmatrix}$ which row-reduces to $\begin{pmatrix} 1 & 0 & -\frac{1}{4} \\ 0 & 1 & \frac{1}{4} \\ 0 & 0 & 0 \end{pmatrix}$.

A unit-length vector in the kernel is $v_2 = \begin{pmatrix} 1/\sqrt{18} \\ -1/\sqrt{18} \\ 4/\sqrt{18} \end{pmatrix}$.

For the last eigenvector, we could compute the kernel of A^TA or find a unit vector perpendicular to v_1 and v_2. To be perpendicular to $v_1 = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ we need $-a = b$. Then the condition that $v_2^T v_3 = 0$ becomes $2a/\sqrt{18} + 4c/\sqrt{18} = 0$ or $-a = 2c$. So $v_3 = \begin{pmatrix} a \\ -a \\ -a/2 \end{pmatrix}$ and for it to be unit-length we need $a = 2/3$ so $v_3 = \begin{pmatrix} 2/3 \\ -2/3 \\ -1/3 \end{pmatrix}$.

How to Calculate SVD

So at this point we know that

$$A = U\Sigma V^T = U \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{pmatrix}.$$

Finally, we can compute U by the formula $\sigma u_i = Av_i$, or $u_i = \frac{1}{\sigma} Av_i$. This gives $U = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$. So in its full glory the SVD is:

$$A = U\Sigma V^T = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{pmatrix}.$$

How to parallelize SVD

- The SVD computation consists of three consecutive steps:
  - (i) bi-diagonalization
  - (ii) computation of singular values and vectors
  - (iii) post-multiplication of results from previous two steps.

One-Sided Block-Jacobi Algorithm (OSBJA)

- Block-column partitioning of A in the form:

$$A = [A_1, A_2, \ldots, A_r]$$

- where the width of Ai is ni , $1 \leq i \leq r$, so that $n1 + n2 + \cdots + nr = n$.

- The OSBJA can be written as an iterative process:

$$A^{(0)} = A, \quad V^{(0)} = I_n,$$
$$A^{(k+1)} = A^{(k)}U^{(k)}, \quad V^{(k+1)} = V^{(k)}U^{(k)}, \quad k \geq 0.$$

One-Sided Block-Jacobi Algorithm (OSBJA)

- The OSBJA can be written as an iterative process:

$$A^{(0)} = A, \quad V^{(0)} = I_n,$$
$$A^{(k+1)} = \boxed{A^{(k)}U^{(k)}}, \quad V^{(k+1)} = \boxed{V^{(k)}U^{(k)}}, \quad k \geq 0. \qquad (1)$$

Single Side

- The purpose of matrix multiplication $A^{(k)}U^{(k)}$ is to mutually orthogonalize the columns between column-blocks i and j of $A^{(k)}$.
- Here the n × n orthogonal matrix $U^{(k)}$ is the so-called block rotation of the form:

$$U^{(k)} = \begin{pmatrix} I & & & \\ & U_{ii}^{(k)} & & U_{ij}^{(k)} \\ & & I & \\ & U_{ji}^{(k)} & & U_{jj}^{(k)} \\ & & & & I \end{pmatrix}, \qquad (2)$$

Jacobi Matrix (Rotation)

One-Sided Block-Jacobi Algorithm (OSBJA)

- The orthogonal matrix

$$\hat{U}^{(k)} = \begin{pmatrix} U_{ii}^{(k)} & U_{ij}^{(k)} \\ U_{ji}^{(k)} & U_{jj}^{(k)} \end{pmatrix} \tag{3}$$

- is called the pivot submatrix of $U^{(k)}$ at step k.

One-Sided Block-Jacobi Algorithm (OSBJA)

- One (serial) step of the OSBJA can be described in three parts:
- 1. For the given pivot pair (i, j), the symmetric, positive semidefinite Gram matrix is computed:

$$\hat{A}_{ij}^{(k)} = [A_i^{(k)} \; A_j^{(k)}]^T [A_i^{(k)} \; A_j^{(k)}] = \begin{pmatrix} A_i^{(k)T} A_i^{(k)} & A_i^{(k)T} A_j^{(k)} \\ A_j^{(k)T} A_i^{(k)} & A_j^{(k)T} A_j^{(k)}. \end{pmatrix} \quad (4)$$

- This requires (ni + nj)(ni + nj −1)/2 dot products
- Or m(ni + nj)(ni + nj −1)/2 flops.
- The two diagonal blocks of $\hat{A}_{ij}^{(k)}$ will be always diagonal. This reduces the flop count to m (ni*nj + ni + nj)
- where m (ni + nj) comes from the computation of the diagonal elements of $\hat{A}^{(k)}$

FLOP (Floating-point operations per second)

One-Sided Block-Jacobi Algorithm (OSBJA)

- 2. $\hat{A}_{ij}^{(k)}$ is diagonalized, i.e., the eigenvalue decomposition of $\hat{A}_{ij}^{(k)}$ is computed:

$$\hat{U}^{(k)T}\,\hat{A}_{ij}^{(k)}\,\hat{U}^{(k)} = \hat{\Lambda}_{ij}^{(k)} \qquad (5)$$

- And the eigenvector matrix $\hat{U}^{(k)}$ is partitioned according to (3). The matrix $\hat{U}^{(k)}$ defines the orthogonal transformation U (k) in (2) and (1), which is then applied to $A^{(k)}$ and $V^{(k)}$.

- Notice that the explicit diagonalization of $\hat{A}^{(k)}$ is equivalent to the implicit mutual orthogonalization of columns between column blocks i and j in $A^{(k)}$, i.e., in ($A_i^{(k)}$, $A_j^{(k)}$). This diagonalization requires on average around $8(ni + nj)^3$ flops.

One-Sided Block-Jacobi Algorithm (OSBJA)

- 3. Finally, an updating of two block-columns of A(k) and V (k) is required, which requires $2m(n_i + n_j)^2$ flops.

- In summary, the kth step of the standard OSBJA requires

$$\begin{aligned} N_{\text{flop}}(k) &\approx m(n_i n_j + n_i + n_j) + 8(n_i + n_j)^3 + 2m(n_i + n_j)^2 \\ &= 64n_0^3 + (9n_0^2 + 2n_0)n \quad (\text{if } m = n = n_0 r) \end{aligned} \tag{6}$$

- Flops.

One-Sided Block-Jacobi Algorithm (OSBJA)

- Each time only 2 column block are involved in the calculation
- Could be implemented with parallel computing

Parallel OSBJA

- Having p processors, the above OSBJA can be parallelized with the blocking factor r = 2p and, for simplicity, assume n1 = n2 = . . . = n2p = n/(2p).
- Hence, each processor contains two block columns and a parallel dynamic ordering has to define which pairs of block columns will meet in a given processor in each parallel iteration step.

Parallel OSBJA

- The computation can be organized in such a way that after the first parallel iteration step (initialization), each block column contains inside orthogonal columns. Let us suppose that all k = n/(2p) columns in each block column are normalized to the unit Euclidean norm.
- Hence, each block column is the orthonormal basis of the k-dimensional subspace which is spanned by the column vectors of a given block column.
- The main idea is to mutually orthogonalize those block columns first which are maximally inclined to each other, i.e., their mutual position differs maximally from the orthogonal one.

Algorithm

Parallel one-sided block-Jacobi SVD algorithm

1: $V = I_n,\ \ell = 2 * p$
2: ▷ each processor has 2 block columns of A : A_L and A_R
3: $G = \begin{pmatrix} G_{LL} & G_{LR} \\ G_{LR}^T & G_{RR} \end{pmatrix} = \begin{pmatrix} A_L^T A_L & A_L^T A_R \\ A_R^T A_L & A_R^T A_R \end{pmatrix}$
4: ▷ *global convergence criterion with a constant* ε, $0 < \varepsilon \ll 1$
5: **while** $(F(A,\ell) \geq \varepsilon)$ **do**
6: ▷ *local convergence criterion with a constant* δ, $0 < \delta \ll 1$
7: **if** $(F(G,\ell) \geq \delta)$ **then**
8: ▷ *diagonalization of* G
9: `EVD`(G,X)
10: ▷ *update of block columns*
11: $(A_L,A_R) = (A_L,A_R) * X$
12: $(V_L,V_R) = (V_L,V_R) * X$
13: **end if**
14: ▷ *parallel ordering–choice of p independent pairs* (i,j) *of block columns*
15: `ReOrderingComp`(p)
16: `Send-Receive`(A_k,V_k,G_{kk}), where k is either L or R
17: **end while**
18: sv_L : square roots of diagonal elements of G_{LL}
19: sv_R : square roots of diagonal elements of G_{RR}
20: ▷ *two block columns of left singular vectors*
21: $U_L = A_L * \mathrm{diag}(1/sv_L),\ U_R = A_R * \mathrm{diag}(1/sv_R)$

EVD(G,X): eigenvalue decompositions of p auxiliary matrices G

Algorithm

- Note that the diagonalization of the auxiliary matrix G is equivalent to the mutual orthogonalization of block columns AL and AR of matrix A.

- Some parallel ordering is required in the procedure ReOrderingComp that defines p independent pairs of block columns of A which are simultaneously mutually orthogonalized in a given parallel iteration step by computing p eigenvalue decompositions EVD(G,X) of p auxiliary matrices G.

Ordering Method

- A big disadvantage of any fixed ordering is the fact that the actual status of orthogonality is usually checked only after a whole sweep and one has no information about the quality of this process at the beginning of a parallel iteration step.
- In other words, in a given parallel iteration step one can try to orthogonalize some mutually ‘almost orthogonal’ block columns while neglecting pairs that are far from being orthogonal.

Ordering Method

- It is clear, at least intuitively, that orthogonalizing block columns with small mutual angles first would mean to eliminate the ‘worst’ pairs first, and this would mean (hopefully) the faster convergence of the whole algorithm as compared with any fixed, cyclic ordering.

- Hence, the main question is how to choose p pairs of block columns with smallest principal angles among all $\ell(\ell-1)/2 = p(2p-1)$ pairs.

Naïve Ordering Method

- The obvious, but very naive way is to compute, for each column block X, all possible matrix products X^TY, then to compute the SVD of X^TY and look at the singular values, which are the cosines of acute principal angles (the smaller angle, the larger cosine)

Naïve Ordering Method

- When the block columns are distributed in processors, to compute matrix products X^TY for each two different block columns X and Y means to move block columns across processors, i.e., it leads to heavy communication at the beginning of each parallel iteration step.
- Besides that, one needs to compute many matrix products and SVDs. Moreover, when p pairs of column blocks with smallest principal angles are chosen, they must meet in processors, which means yet another communication.

Dynamic Ordering

- After the first parallel iteration step, the block columns inside contain mutually orthogonal columns.
- Suppose that each processor contains exactly two block columns (this is not substantial for the following discussion).
- Moreover, suppose that $k \equiv n/2p$ columns in each block column are normalized so that each has the unit Euclidean norm.
- Hence, each column block is the orthonormal basis of the k-dimensional subspace which is spanned by the column vectors of a given block column.

Dynamic Ordering

- Having p processors, our goal is to choose p pairs of those block columns that are maximally inclined to each other,
- i.e., their mutual position differs maximally from the orthogonal one.
- Mathematically, this goal can be described by using the notion of principal angles between two k-dimensional subspaces spanned by two block columns Ai , Aj .
- We are interested in, say, L smallest principal angles, i.e., in L largest cosines (largest singular values)

Dynamic Ordering

- To estimate L largest singular value of the k × k matrix $A^T_i A_j$, use the Lanczos process applied to the symmetric Jordan-Wielandt matrix C

$$C \equiv \begin{pmatrix} 0 & A_i^T A_j \\ A_j^T A_i & 0 \end{pmatrix}.$$

- It is well known that the eigenvalues of the 2k ×2k matrix C has k pairs of eigenvalues with the same absolute value.

- Use the Approximated values as weight to rank all the column blocks.

TABLE 2.1
Performance for $n = 4000,\ p = 8,\ \kappa = 10^1$

mode		$L=1$	$L=2$	$L=4$	$L=6$	static1	static2
1	n_{it}	4	4	4	4	30	30
	T_p [s]	5	6	9	11	13	13
2	n_{it}	4	4	4	4	30	30
	T_p [s]	5	6	9	11	13	12
3	n_{it}	108	99	98	99	240	270
	T_p [s]	225	234	292	354	354	390
4	n_{it}	103	97	98	97	225	240
	T_p [s]	212	228	289	345	327	354
5	n_{it}	109	103	99	100	255	285
	T_p [s]	230	242	293	360	367	409
6	n_{it}	108	107	106	103	270	285
	T_p [s]	226	252	315	371	395	420

- L is # of iterations in approximation, order n=4000 matrix
- static1 (the odd-even ordering CO(0))and static2 (the robinround ordering DO(0))

Future Work

- Literature Survey on more possible solution
- Implement Algorithm with MPI