Assignment (EE664)

Report by

150108001: Afzal Ahmad
150102065: Shubham Tulasyan
150107039: Aditya Babu

Course Instructor

Dr. Gaurav Trivedi

DEPARTMENT OF ELECTRONICS & ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Contents

1 Abstract

This report describes the synopsis of the serial implementation of the algorithm presented by paper Zhong-Zhi Bai[1]. We give a brief introduction of the problem in section 2 which is followed by the algorithm in section 3. We present our results for the serial implementation in section 3.

2 Introduction

A linear complimnetairty problem, abbreviated as LCP(A,q) can be defined as where A

$$w := Az + q \geq 0, \quad z \geq 0 \quad and\ z^T w = 0$$

denotes a large, real and sparse matrix and q is a real vector. The $\geq$ in the equation denotes a component wise defined partial ordering between the two vectors. The LCP aims to find a pair of z and w such that the above equations are satisfied.

Linear complimentarty problems are a very important in the field of optimisation problems. They also find use in scientific computing and several engineering applications. For ex. the free boundary problem and other problems in computational mechanics.

In many problems, the matrix A can be very large which makes it difficult to compute a numerical solution for the problem. Several matrix splitting based iterative methods have been proposed to efficiently solve LCP(A,q) for large A. One such solution is to reformulate the LCP(A,q) as an implicit fixed point equation. This method was explored presented as a modulus iteration methods. An iteration parameter is used to reduce the number of iterations required for convergence.

The given paper introduces a modulus-based matrix splitting iteration methods that takes advantage of the matrix splitting to solve for large sparse matrix A. It covers the previous modulus iteration method and its variants and also introduces a new class of modulus-based relaxation methods.

3 Algorithm

The LCP(A,q) can be reformulated into the following implicit fixed point equation.

$$(\Omega + M)x = (\Omega - A)|x| + Nx - \gamma q \tag{1}$$

So,the corresponding iterative equation can be written as,

$$(\Omega + M)x^{(k+1)} = (\Omega - A)|x^k| + Nx^k - \gamma q \tag{2}$$

Thus, by splitting matrix A as A=M - N and using an initial $x^{(0)}$ vector, the above equation (3) is solved for iteration k = 0,1,2,3 . .. until the sequence x^k converges. The solutions of the LCP(A,q) are then given by,

$$z^{(k+1)} = 1/\gamma(x^{(k+1)} + |x^{(k+1)}|), \quad w^{(k+1)} = \Omega(|x^{(k+1)}| - x^{(k+1)})$$

Here,Ω is an $n \times n$ positive diagonal matrix and γ is a positive constant.

Equation (2) stated above, provides a generalized framework of modulus based splitting iteration method for matrices. For different splitting of matrix A, whether it be A=M - N or A=D-L-U where D,L,U are diagonal, strictly lower-triangular and strictly upper-triangular matrices of matrix A respectively and different iteration parameters, a series of modulus based iteration methods are developed which are described below.

3.1 Modified Modulus Iteration Method

For $N = 0$, $M = A$, $\gamma = 1$ *and* $\Omega = \alpha I$, equation (2) gets reduced to

$$(\alpha I + A)x^{(k+1)} = (\alpha I - A)|x^k| - q \tag{3}$$

$$z^{(k+1)} = (x^{(k+1)} + |x^{(k+1)}|), \quad w^{(k+1)} = \alpha I(|x^{(k+1)}| - x^{(k+1)}) \tag{4}$$

3.2 Modulus Based Gauss Seidel (MGS) Iteration Method

For $N = U$, $M = D$ - L *and* $\gamma = 2$, equation (2) gets reduced to

$$(D - L + \Omega)x^{(k+1)} = Ux^k + (\Omega - A)|x^k| - 2q \tag{5}$$

$$z^{(k+1)} = 1/2(x^{(k+1)} + |x^{(k+1)}|), \quad w^{(k+1)} = \Omega(|x^{(k+1)}| - x^{(k+1)}) \tag{6}$$

3.3 Modulus Based Successive Overrelaxation (MSOR) Iteration Method

For $N = (1/\alpha - 1)D + U, M = (1/\alpha)D - L$ *and* $\gamma = 2$, equation (2) gets reduced to

$$(\Omega + D - \alpha L)x^{(k+1)} = [\alpha U + (1 - \alpha)D]x^k - 2\alpha q + (\Omega - \alpha A)|x^k| \tag{7}$$

$$z^{(k+1)} = 1/2(x^{(k+1)} + |x^{(k+1)}|), \quad w^{(k+1)} = \Omega(|x^{(k+1)}| - x^{(k+1)}) \tag{8}$$

4 Numerical Results for Serial Implementation

The algorithms described above were implemented in C++ programming language in a serial manner. *'RES'* was used a the convergence criteria. *'RES'* is defined as

$$RES(z^k) := ||min(Az^k + q, z^k)||_2 \tag{9}$$

All the iterations were terminated once the $RES(z^k) \leq 10^{-5}$.

The initial vector used for all the computations is

$$x^{(0)} = (1, 0, 1, 0, ..., 1, 0, ...)^T \in \mathbb{R}^n \tag{10}$$

We use $\Omega = (1/(2\alpha))D$ for the MGS and MSOR methods. The parameter α is used to accelerate the rate of convergence and minimize the number of iterations.

To test the presented algorithms,we use predefined inputs A and q and solve the LCP(A,q) to converge at a specific value of z vector. The input $A = \hat{A} + \mu I$ and $q \in \mathbb{R}^n$ is a vector defined as $q = -Mz_{(0)}$, where

$$z^{(0)} = (1, 2, 1, 2, ..., 1, 2, ...)^T \in \mathbb{R}^n \tag{11}$$

and the block matrix S is given by,

$$\widehat{A} = \text{Tridiag}(-I, S, -I) = \begin{pmatrix} S & -I & 0 & \cdots & 0 & 0 \\ -I & S & -I & \cdots & 0 & 0 \\ 0 & -I & S & \cdots & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & S & -I \\ 0 & 0 & \cdots & \cdots & -I & S \end{pmatrix} \in \mathbb{R}^{n \times n}$$

$$S = \text{tridiag}(-1, 4, -1) = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 4 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 4 & \cdots & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & 4 & -1 \\ 0 & 0 & \cdots & \cdots & -1 & 4 \end{pmatrix} \in \mathbb{R}^{m \times m}$$

The $z_{(0)}$ described above is the unique solution of the LCP(A,q).

The results for MM, MGS and MSOR algorithms are shown below for different input size matrix (n= 100, 400 and 900). Each figure shows CPU execution time and no. of iteration varying with different values of α. α is varied to reduce the no. of iterations.

4.1 Results for MM Serial Implementation

For MM algorithm, the omega matrix $\Omega = \alpha I$ and for every value of input matrix size (n), the no of iteration first decreases and then increases with α. The results have been simulated for different values of alpha from 4.0 to 9.0 to find alpha with optimal iteration which turns out to be 6.5 in all the above cases.

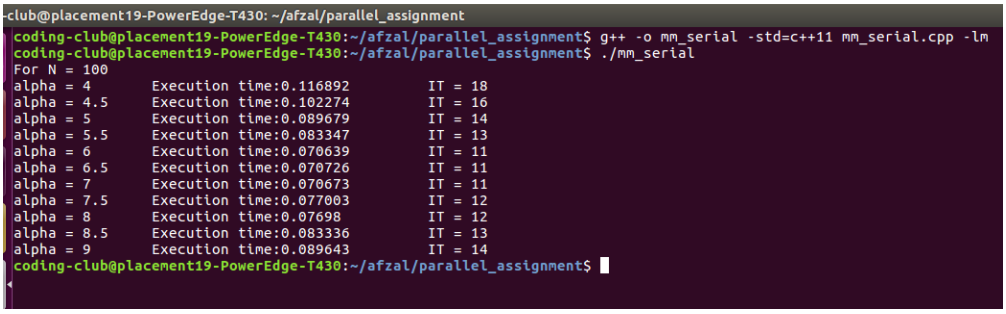

Figure 1: n=100

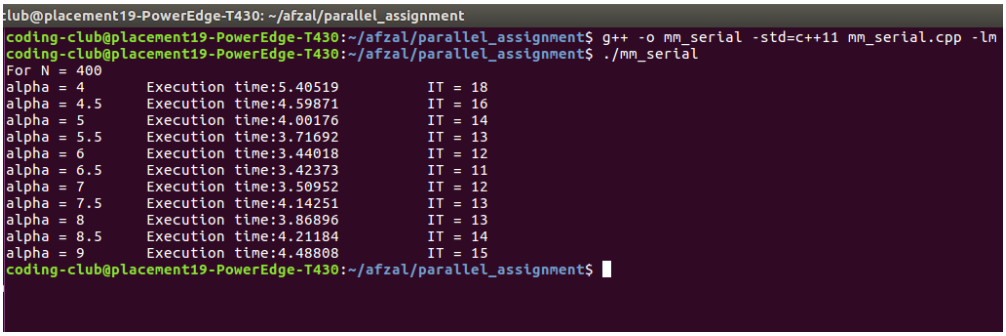

Figure 2: n=400

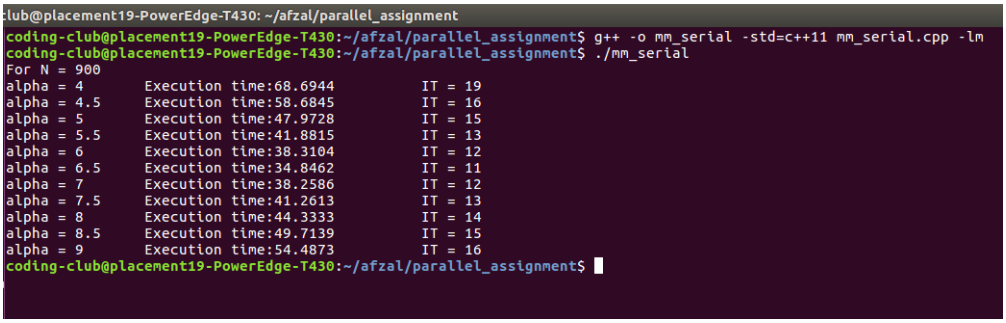

Figure 3: n=900

4.2 Results for MGS Serial Implementation

For MGS algorithm, the omega matrix $\Omega = (1/2\alpha)$D where D is a diagonal matrix. For different input matrix size (n), the results have been simulated for different values of alpha from 0.4 to 1.0 to find alpha with optimal iteration which turns out to be 0.7 in all the above cases.

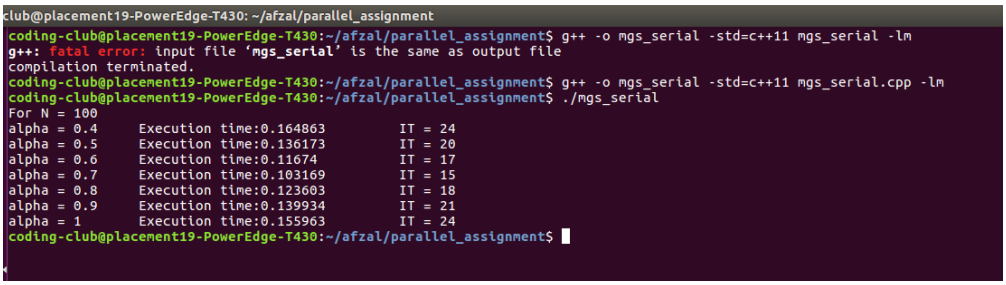

Figure 4: n=100

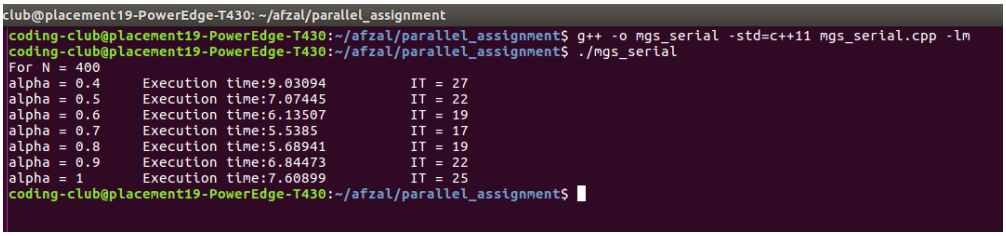

Figure 5: n=400

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mgs_serial -std=c++11 mgs_serial.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mgs_serial
For N = 900
alpha = 0.4     Execution time:91.1004          IT = 28
alpha = 0.5     Execution time:73.7882          IT = 23
alpha = 0.6     Execution time:64.2819          IT = 20
alpha = 0.7     Execution time:56.6949          IT = 17
alpha = 0.8     Execution time:61.411           IT = 19
alpha = 0.9     Execution time:77.6005          IT = 22
alpha = 1       Execution time:90.112           IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 6: n=900

4.3 Results for MSOR Serial Implementation

For MGS algorithm, the omega matrix $\Omega = (1/2\alpha)$D where D is a diagonal matrix. For different input matrix size (n), the results have been simulated for different values of alpha from 0.4 to 1.0 to find alpha with optimal iteration which turns out to be 0.9 in all the above cases.

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_s -std=c++11 msor_serial.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_s
For N = 100
alpha = 0.4     Execution time:0.503574         IT = 83
alpha = 0.5     Execution time:0.330759         IT = 55
alpha = 0.6     Execution time:0.234532         IT = 39
alpha = 0.7     Execution time:0.168461         IT = 28
alpha = 0.8     Execution time:0.126359         IT = 21
alpha = 0.9     Execution time:0.090245         IT = 15
alpha = 1       Execution time:0.144029         IT = 24
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 7: n=100

```
Terminal  File  Edit  View  Search  Terminal  Help
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_s -std=c++11 msor_serial.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_s
For N = 400
alpha = 0.4     Execution time:27.3079          IT = 92
alpha = 0.5     Execution time:18.0793          IT = 61
alpha = 0.6     Execution time:13.6143          IT = 43
alpha = 0.7     Execution time:9.15331          IT = 31
alpha = 0.8     Execution time:6.81273          IT = 23
alpha = 0.9     Execution time:5.03259          IT = 17
alpha = 1       Execution time:7.69374          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 8: n=400

```
Terminal  File  Edit  View  Search  Terminal  Help
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_s -std=c++11 msor_serial.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_s
For N = 900
alpha = 0.4     Execution time:317.132          IT = 95
alpha = 0.5     Execution time:212.956          IT = 63
alpha = 0.6     Execution time:145.032          IT = 45
alpha = 0.7     Execution time:106.802          IT = 33
alpha = 0.8     Execution time:82.1116          IT = 24
alpha = 0.9     Execution time:58.5344          IT = 18
alpha = 1       Execution time:81.0671          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 9: n=900

The above results for serial implementation for all 3 algorithms is tabulated below showing no. of iterations and CPU execution time for optimal value of α.

Numerical Results for $\mu = 4$									
	n=100			**n=400**			**n=900**		
Method	α	CPU	IT	α	CPU	IT	α	CPU	IT
MSOR	0.9	0.0902	15	0.9	5.0325	17	0.9	58.5344	18
MGS	0.7	0.103169	15	0.7	5.5385	17	0.7	56.9649	17
MM	6.5	0.07072	11	6.5	3.4237	11	6.5	34.4862	11

5 Numerical Results for Parallel Implementation

To further improve the CPU execution time, we have identified the scope of parallelization in previously implemented serial code for MM, MGS and MSOR methods and parallelized the identified section using OpenMP library accordingly. Since we are implementing an iterative algorithm, the scope of parallelization is limited and only various matrix operation like matrix addition, subtraction can be parallelized. Further, the bulk of CPU execution time is limited by finding solution of linear equation $Ax = b$ which forms the basis of our iterative algorithm. We have used Cholesky decomposition method for solving linear equation and reduced execution time by parallelizing code section wherever possible. The numerical results for inputs previously described is shown below for various matrix sizes and no of available threads (4,8,16,32).

5.1 Results For MM Parallel Implementation

For matrix size $n = 100$

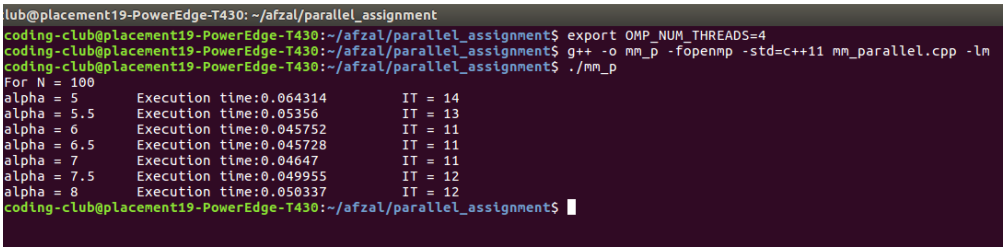

Figure 10: No. of threads = 4 and n=100

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=8
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 100
alpha = 5       Execution time:0.0666           IT = 14
alpha = 5.5     Execution time:0.057875         IT = 13
alpha = 6       Execution time:0.048719         IT = 11
alpha = 6.5     Execution time:0.049338         IT = 11
alpha = 7       Execution time:0.048863         IT = 11
alpha = 7.5     Execution time:0.053639         IT = 12
alpha = 8       Execution time:0.053496         IT = 12
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 11: No. of threads = 8 and n=100

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=16
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 100
alpha = 5       Execution time:0.103277         IT = 14
alpha = 5.5     Execution time:0.092105         IT = 13
alpha = 6       Execution time:0.078976         IT = 11
alpha = 6.5     Execution time:0.078608         IT = 11
alpha = 7       Execution time:0.078941         IT = 11
alpha = 7.5     Execution time:0.08576          IT = 12
alpha = 8       Execution time:0.0854           IT = 12
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 12: No. of threads = 16 and n=100

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=32
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 100
alpha = 5       Execution time:0.125556         IT = 14
alpha = 5.5     Execution time:0.111771         IT = 13
alpha = 6       Execution time:0.095306         IT = 11
alpha = 6.5     Execution time:0.094741         IT = 11
alpha = 7       Execution time:0.09528          IT = 11
alpha = 7.5     Execution time:0.106638         IT = 12
alpha = 8       Execution time:0.103831         IT = 12
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 13: No. of threads = 32 and n=100

For matrix size $n = 400$

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=4
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 400
alpha = 5       Execution time:1.69585          IT = 14
alpha = 5.5     Execution time:1.49086          IT = 13
alpha = 6       Execution time:1.35366          IT = 12
alpha = 6.5     Execution time:1.3055           IT = 11
alpha = 7       Execution time:1.38086          IT = 12
alpha = 7.5     Execution time:1.54748          IT = 13
alpha = 8       Execution time:1.52867          IT = 13
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 14: No. of threads = 4 and n=400

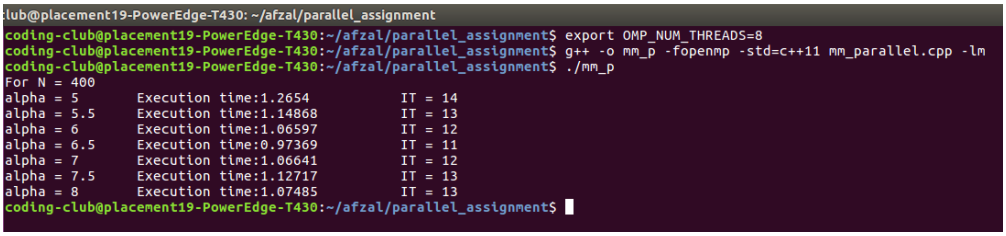

Figure 15: No. of threads = 8 and n=400

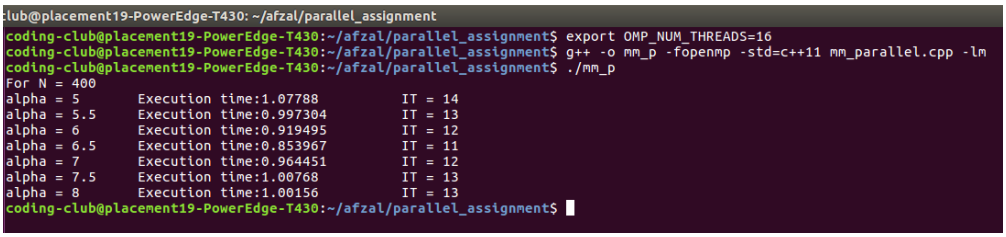

Figure 16: No. of threads = 16 and n=400

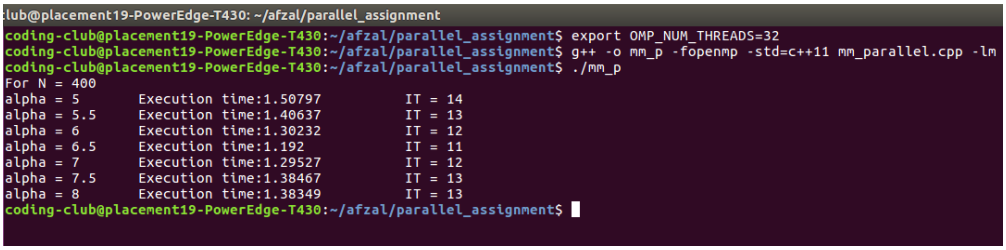

Figure 17: No. of threads = 32 and n=400

For matrix size $n = 900$

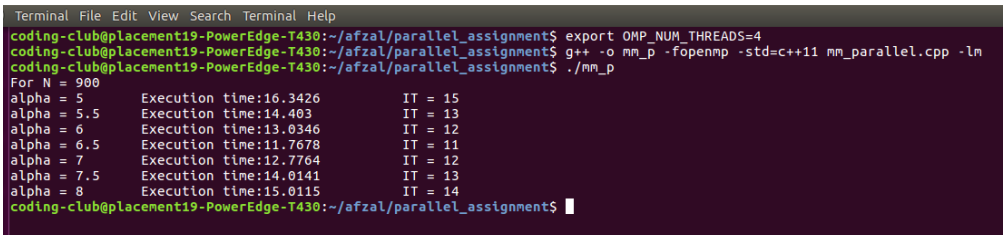

Figure 18: No. of threads = 4 and n=900

```
Terminal  File  Edit  View  Search  Terminal  Help
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=8
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 900
alpha = 5       Execution time:10.184           IT = 15
alpha = 5.5     Execution time:8.84763          IT = 13
alpha = 6       Execution time:8.0051           IT = 12
alpha = 6.5     Execution time:7.43065          IT = 11
alpha = 7       Execution time:7.93856          IT = 12
alpha = 7.5     Execution time:8.64498          IT = 13
alpha = 8       Execution time:9.23635          IT = 14
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 19: No. of threads = 8 and n=900

```
Terminal  File  Edit  View  Search  Terminal  Help
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=16
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 900
alpha = 5       Execution time:8.33777          IT = 15
alpha = 5.5     Execution time:7.39586          IT = 13
alpha = 6       Execution time:7.2545           IT = 12
alpha = 6.5     Execution time:7.25644          IT = 11
alpha = 7       Execution time:6.59797          IT = 12
alpha = 7.5     Execution time:7.02226          IT = 13
alpha = 8       Execution time:7.62218          IT = 14
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 20: No. of threads = 16 and n=900

```
Terminal  File  Edit  View  Search  Terminal  Help
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=32
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mm_p -fopenmp -std=c++11 mm_parallel.cpp -lm
^[[Acoding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mm_p
For N = 900
alpha = 5       Execution time:8.40669          IT = 15
alpha = 5.5     Execution time:7.42725          IT = 13
alpha = 6       Execution time:6.83519          IT = 12
alpha = 6.5     Execution time:6.37359          IT = 11
alpha = 7       Execution time:6.81059          IT = 12
alpha = 7.5     Execution time:7.37643          IT = 13
alpha = 8       Execution time:7.93068          IT = 14
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 21: No. of threads = 32 and n=900

As observed in above simulation results, when input size is small (n =100) addition of extra threads does not have intended effect of reducing execution time, further it is increased with addition of CPU core. But when input size is increased (n = 400, 900), CPU execution time reduces with addition of threads. Also there is decrease in execution time as compared with serial implementation with most of time execution time gets halved and in case of large matrix (n =900), reduction is 5 times. The above results for MM parallel implementation for different input size and no. of threads is tabulated below showing no. of iterations and CPU execution time for optimal value of α.

Numerical Results for $\mu = 4$									
	n=100			**n=400**			**n=900**		
Threads	α	CPU	IT	α	CPU	IT	α	CPU	IT
4	6.5	0.045728	11	6.5	1.3055	11	6.5	11.7678	11
8	7.0	0.048863	11	6.5	0.97369	11	6.5	7.43065	11
16	6.5	0.078608	11	6.5	0.8539	11	6.5	7.25644	11
32	6.5	0.094741	11	6.5	1.192	11	6.5	6.3735	11

5.2 Results For MGS Parallel Implementation

For matrix size $n = 100$

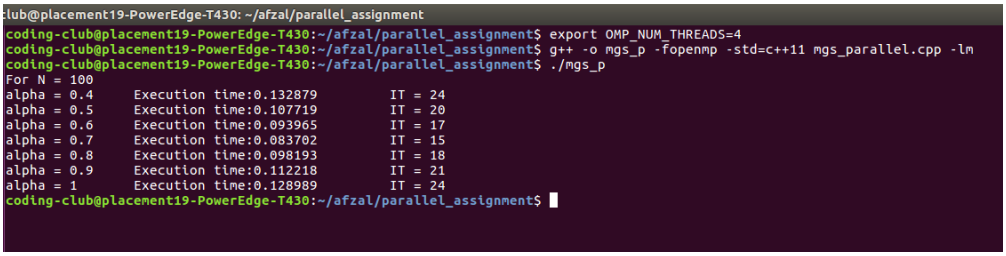

Figure 22: No. of threads = 4 and n=100

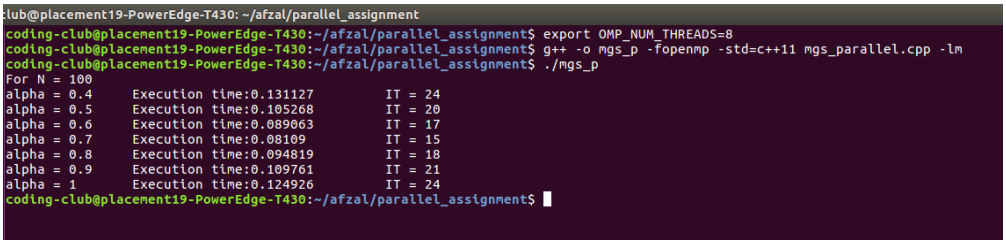

Figure 23: No. of threads = 8 and n=100

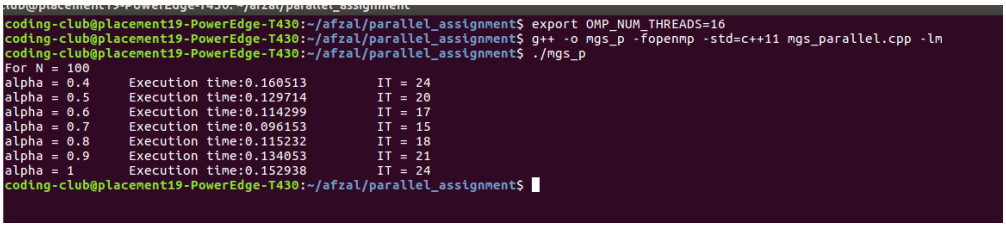

Figure 24: No. of threads = 16 and n=100

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=32
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mgs_p -fopenmp -std=c++11 mgs_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mgs_p
For N = 100
alpha = 0.4     Execution time:0.22143          IT = 24
alpha = 0.5     Execution time:0.181389         IT = 20
alpha = 0.6     Execution time:0.156884         IT = 17
alpha = 0.7     Execution time:0.13899          IT = 15
alpha = 0.8     Execution time:0.166032         IT = 18
alpha = 0.9     Execution time:0.187055         IT = 21
alpha = 1       Execution time:0.216118         IT = 24
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 25: No. of threads = 32 and n=100

For matrix size $n = 400$

```
Terminal  File  Edit  View  Search  Terminal  Help
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=4
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mgs_p -fopenmp -std=c++11 mgs_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mgs_p
For N = 400
alpha = 0.4     Execution time:3.10719          IT = 27
alpha = 0.5     Execution time:2.59522          IT = 22
alpha = 0.6     Execution time:2.2488           IT = 19
alpha = 0.7     Execution time:2.0375           IT = 17
alpha = 0.8     Execution time:2.16724          IT = 19
alpha = 0.9     Execution time:2.50832          IT = 22
alpha = 1       Execution time:2.88379          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 26: No. of threads = 4 and n=400

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=8
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mgs_p -fopenmp -std=c++11 mgs_parallel.cpp -lm
^[[Acoding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mgs_p
For N = 400
alpha = 0.4     Execution time:2.3693           IT = 27
alpha = 0.5     Execution time:1.94764          IT = 22
alpha = 0.6     Execution time:1.71258          IT = 19
alpha = 0.7     Execution time:1.65939          IT = 17
alpha = 0.8     Execution time:1.72671          IT = 19
alpha = 0.9     Execution time:1.98476          IT = 22
alpha = 1       Execution time:2.25461          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 27: No. of threads = 8 and n=400

```
club@placement19-PowerEdge-T430: ~/afzal/parallel_assignment
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=16
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o mgs_p -fopenmp -std=c++11 mgs_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./mgs_p
For N = 400
alpha = 0.4     Execution time:2.59959          IT = 27
alpha = 0.5     Execution time:2.10148          IT = 22
alpha = 0.6     Execution time:1.80385          IT = 19
alpha = 0.7     Execution time:1.66227          IT = 17
alpha = 0.8     Execution time:1.82647          IT = 19
alpha = 0.9     Execution time:2.09996          IT = 22
alpha = 1       Execution time:2.40692          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 28: No. of threads = 16 and n=400

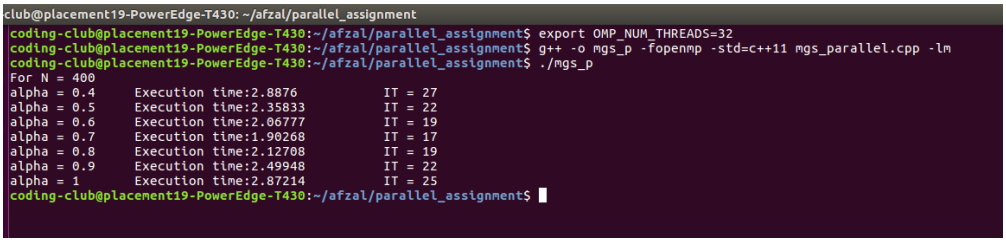

Figure 29: No. of threads = 32 and n=400

For matrix size $n = 900$

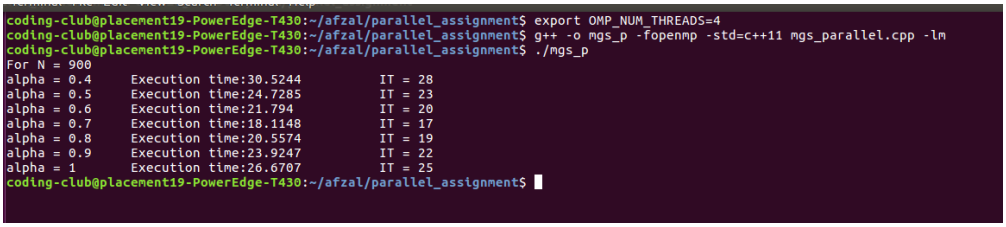

Figure 30: No. of threads = 4 and n=900

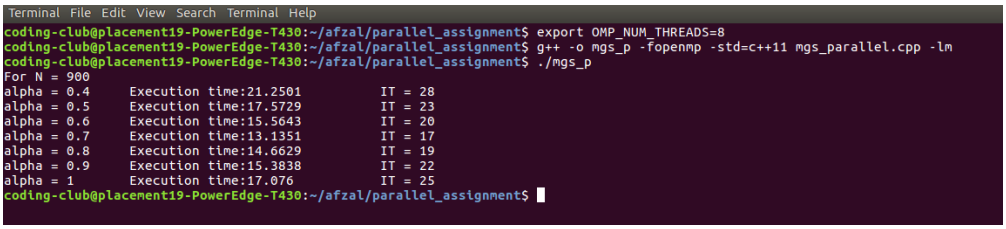

Figure 31: No. of threads = 8 and n=900

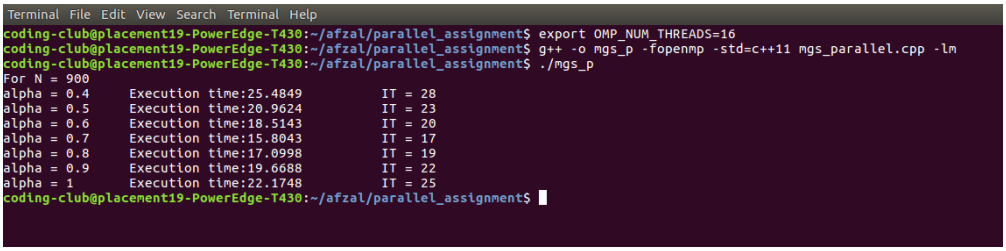

Figure 32: No. of threads = 16 and n=900

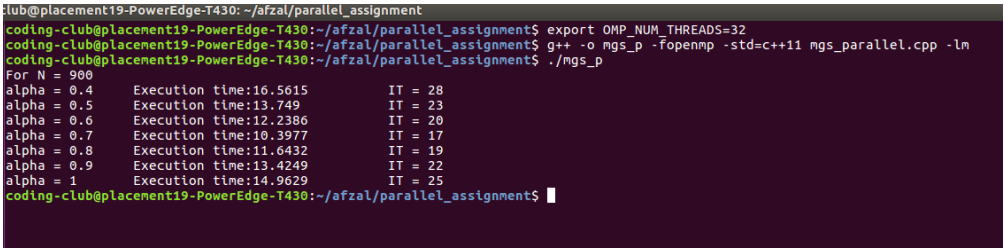

Figure 33: No. of threads = 32 and n=900

As observed in above simulation results, when input size is small (n =100) addition of extra threads does not have intended effect of reducing execution time, further it is increased with addition of CPU core. But when input size is increased (n = 400, 900), CPU execution time reduces with addition of threads. Also there is decrease in execution time as compared with serial implementation with most of time execution time gets halved and in case of large matrix (n =900), reduction is 5 times. The above results for MGS parallel implementation for different input size and no. of threads is tabulated below showing no. of iterations and CPU execution time for optimal value of α.

Numerical Results for $\mu = 4$

	n=100			**n=400**			**n=900**		
Threads	α	CPU	IT	α	CPU	IT	α	CPU	IT
4	0.7	0.083702	15	0.7	2.0375	17	0.7	18.1148	17
8	0.7	0.08109	15	0.7	1.65939	17	0.7	13.1351	17
16	0.7	0.096153	15	0.7	1.66227	17	0.7	15.8043	17
32	0.7	0.13899	15	0.7	1.90268	17	0.7	10.3977	17

5.3 Results For MSOR Parallel Implementation

For matrix size $n = 100$

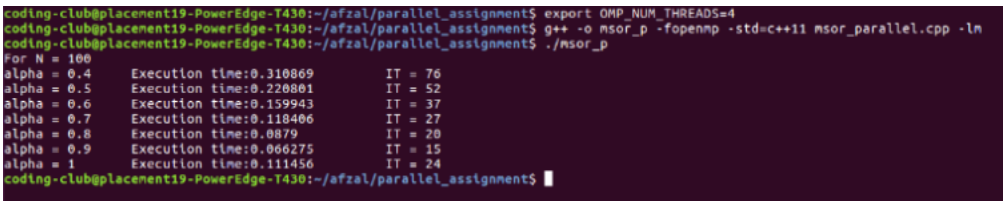

Figure 34: No. of threads = 4 and n=100

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=8
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 100
alpha = 0.4     Execution time:0.385991         IT = 76
alpha = 0.5     Execution time:0.261512         IT = 52
alpha = 0.6     Execution time:0.185302         IT = 37
alpha = 0.7     Execution time:0.132297         IT = 27
alpha = 0.8     Execution time:0.094622         IT = 20
alpha = 0.9     Execution time:0.070758         IT = 15
alpha = 1       Execution time:0.117122         IT = 24
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 35: No. of threads = 8 and n=100

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=16
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 100
alpha = 0.4     Execution time:0.590378         IT = 76
alpha = 0.5     Execution time:0.399155         IT = 52
alpha = 0.6     Execution time:0.283464         IT = 37
alpha = 0.7     Execution time:0.206937         IT = 27
alpha = 0.8     Execution time:0.152572         IT = 20
alpha = 0.9     Execution time:0.114468         IT = 15
alpha = 1       Execution time:0.183993         IT = 24
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 36: No. of threads = 16 and n=100

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=32
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 100
alpha = 0.4     Execution time:0.713157         IT = 76
alpha = 0.5     Execution time:0.479401         IT = 52
alpha = 0.6     Execution time:0.340248         IT = 37
alpha = 0.7     Execution time:0.249274         IT = 27
alpha = 0.8     Execution time:0.184469         IT = 20
alpha = 0.9     Execution time:0.138207         IT = 15
alpha = 1       Execution time:0.220407         IT = 24
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 37: No. of threads = 32 and n=100

For matrix size $n = 400$

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=4
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 400
alpha = 0.4     Execution time:10.639           IT = 84
alpha = 0.5     Execution time:7.2396           IT = 57
alpha = 0.6     Execution time:5.21726          IT = 41
alpha = 0.7     Execution time:3.7949           IT = 30
alpha = 0.8     Execution time:2.92268          IT = 23
alpha = 0.9     Execution time:2.15123          IT = 17
alpha = 1       Execution time:3.19754          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 38: No. of threads = 4 and n=400

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=8
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 400
alpha = 0.4     Execution time:8.31431          IT = 84
alpha = 0.5     Execution time:6.41964          IT = 57
alpha = 0.6     Execution time:4.66497          IT = 41
alpha = 0.7     Execution time:3.56549          IT = 30
alpha = 0.8     Execution time:2.63206          IT = 23
alpha = 0.9     Execution time:1.9277           IT = 17
alpha = 1       Execution time:3.0157           IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 39: No. of threads = 8 and n=400

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=16
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 400
alpha = 0.4     Execution time:7.84451          IT = 84
alpha = 0.5     Execution time:5.59325          IT = 57
alpha = 0.6     Execution time:3.95857          IT = 41
alpha = 0.7     Execution time:2.82135          IT = 30
alpha = 0.8     Execution time:2.17635          IT = 23
alpha = 0.9     Execution time:1.62699          IT = 17
alpha = 1       Execution time:2.38882          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 40: No. of threads = 16 and n=400

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=32
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 400
alpha = 0.4     Execution time:10.1371          IT = 84
alpha = 0.5     Execution time:6.88954          IT = 57
alpha = 0.6     Execution time:4.95828          IT = 41
alpha = 0.7     Execution time:3.63598          IT = 30
alpha = 0.8     Execution time:2.82292          IT = 23
alpha = 0.9     Execution time:2.09411          IT = 17
alpha = 1       Execution time:3.02812          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 41: No. of threads = 32 and n=400

For matrix size $n = 900$

```
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_N
UM_THREADS=4
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_
p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 900
alpha = 0.4     Execution time:92.0263          IT = 88
alpha = 0.5     Execution time:66.6752          IT = 60
alpha = 0.6     Execution time:46.1411          IT = 43
alpha = 0.7     Execution time:31.1653          IT = 31
alpha = 0.8     Execution time:22.7268          IT = 23
alpha = 0.9     Execution time:16.8054          IT = 17
alpha = 1       Execution time:25.3469          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 42: No. of threads = 4 and n=900

```
coding-club@placement19-PowerEdge-T430:~$ cd afzal/parallel_assignment/
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ export OMP_NUM_THREADS=8
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ g++ -o msor_p -fopenmp -std=c++11 msor_parallel.cpp -lm
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$ ./msor_p
For N = 900
alpha = 0.4     Execution time:68.6477          IT = 88
alpha = 0.5     Execution time:46.9436          IT = 60
alpha = 0.6     Execution time:33.6216          IT = 43
alpha = 0.7     Execution time:24.2706          IT = 31
alpha = 0.8     Execution time:18.003           IT = 23
alpha = 0.9     Execution time:13.3233          IT = 17
alpha = 1       Execution time:19.7031          IT = 25
coding-club@placement19-PowerEdge-T430:~/afzal/parallel_assignment$
```

Figure 43: No. of threads = 8 and n=900

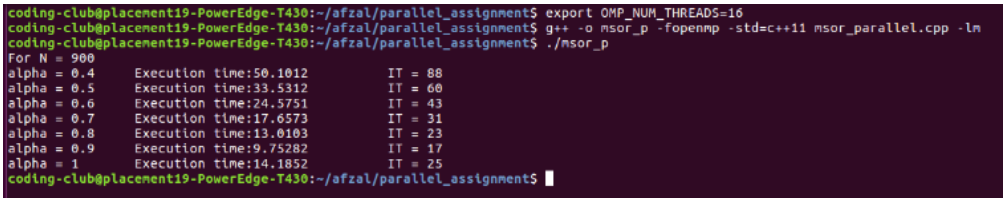

Figure 44: No. of threads = 16 and n=900

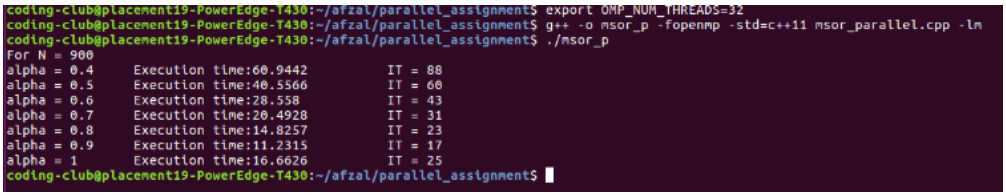

Figure 45: No. of threads = 32 and n=900

As observed in above simulation results, when input size is small (n =100) addition of extra threads does not have intended effect of reducing execution time, further it is increased with addition of CPU core. But when input size is increased (n = 400,900), CPU execution time first reduces with addition of threads then increases. Also there is decrease in execution time as compared with serial implementation with most of time execution time gets halved and in case of large matrix (n =900), reduction is 5 times. The above results for MSOR parallel implementation for different input size and no. of threads is tabulated below showing no. of iterations and CPU execution time for optimal value of α.

Numerical Results for $\mu = 4$

	n=100			**n=400**			**n=900**		
Threads	α	CPU	IT	α	CPU	IT	α	CPU	IT
4	0.9	0.066275	15	0.9	2.15123	17	0.9	16.8054	17
8	0.9	0.070758	15	0.9	1.9277	17	0.9	13.3233	17
16	0.9	0.114468	15	0.9	1.62699	17	0.9	9.75282	17
32	0.9	0.138207	15	0.9	2.09411	17	0.9	11.2315	17

References

[1] Bai, Z. (2010), ModulusâĂŘbased matrix splitting iteration methods for linear complementarity problems. Numer. Linear Algebra Appl., 17: 917-933. doi:10.1002/nla.680