CAAM520 Computational Science

Homework 2

AO CAI
S01255664
Earth Science Department
Rice University
Feb 17, 2019

Contents

1 Introduction

The Goal is to solve the matrix system $Au = b$ resulting from an $(N+2) \times (N+2)$ 2D finite difference method for Laplace's equation $-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x, y)$ on $[-1, 1]^2$. At each point (x_i, y_i), derivatives are approximated by:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

where $h = 2/(N+1)$. Assuming zero boundary conditions $u(x, y) = 0$ reduce this to an $N \times N$ system for the interior nodes.

2 MPI Implementation

To implement the parallel computing for the Poisson's equaion, I first divide the matrix into blocks based on the number of processors are used. When the total problem is of size $N \times N$ (without the zero-boundaries), I distributed the original wavefield into different processors. The size of the new wavefield is:

$$sizeof(u) = N * nub * sizeof(double)$$

where $nub = node + 2$ if the rank of processor is less than $size - 1$, or $nub = node + res + 2$ for the last processor $rank = size - 1$. The node is the integer of $\frac{N}{size}$ and nres is the residual of $mod(N, size)$. Then, I am able to allocate the memory of u in the different processor evenly. The extra 2 layer in each processor is used to communicate the information between each processors. Since after each iteration, the wavefield in the top layer and bottom layer should be updated based on the information from other processors. The FD operations in each layer is the same:

$$-Ru \approx \frac{-u_{i,j+1} - u_{i+1,j} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

The after this computing, the wavefield on each node is updated with

$$u^{(k+1)} = \omega D^{-1}(b - Ru^{(k)}) + (1 - \omega)u^{(k)}$$

The wavefield is updated in the internal grids from index 0 to N in x direction and from 1 to $nub - 1$ in the z direction.

Figure 1 in the following shows how the parallelism functions in the $MPIJacob.c$. Every processor has a memory block of u with size $u[N][nub]$. The shadow area is the position of active variable, each of them represent part of the total wavefield u. The last layer on processer k-1 serves as the bottom boundary, we want it to communicate with the layer 1 in the processor k (not the top boundary). Besides, in the processor k, the top boundary needs to be updated base on the value of layer $nub-1$ in the processor k-1.The psudocode is working as (psudocode):

$$while(obj >= tol, iter < itermax)$$

update the wavefield on each processors based on

$$u^{(k+1)} = \omega D^{-1}(b - Ru^{(k)}) + (1-\omega)u^{(k)}$$

After that:

Exchange the wavefield boundaries in different layers, send layer 1 in proc k to the bottom boundary in proc k-1, and send layer $nub-1$ in proc k to the top boundary in proc k+1

Use MPI Reduce to calculate the global objective funcion obj from localobj on different processors.

Use MPI Bcast to broadcast the obj value into different processor

$$iter = iter + 1$$

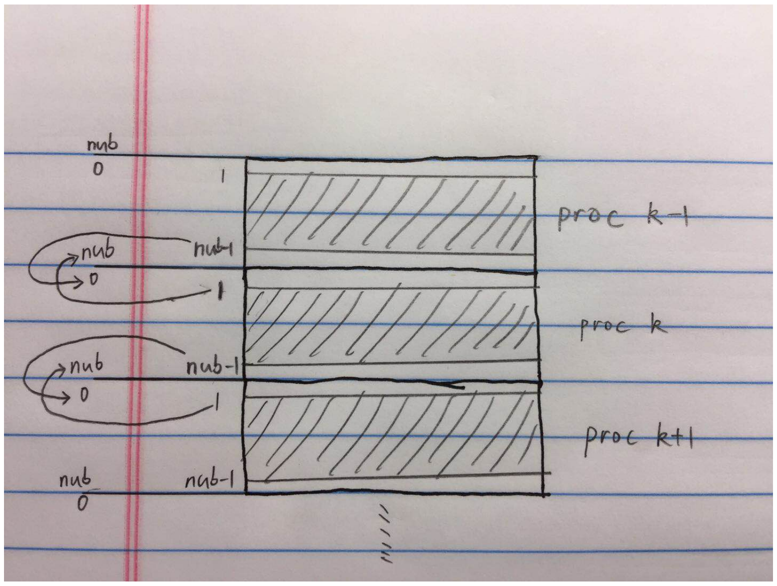


Figure 1: Sample plot of the communications between the layers

The difficulty comes from that there are two communication pairs between proc k and proc k-1, and between proc k and proc k+1. If we use the same order of send and receive, we will come to the problem of deadlock.

To avoid this problem, I use different order of send and receive in the even and odd ranks.

If $mod(rank, 2) == 1)$

I will first do MPI Send and then MPI Recv.

If $mod(rank, 2) == 0$

I will first do MPI Recv and then MPI Send.

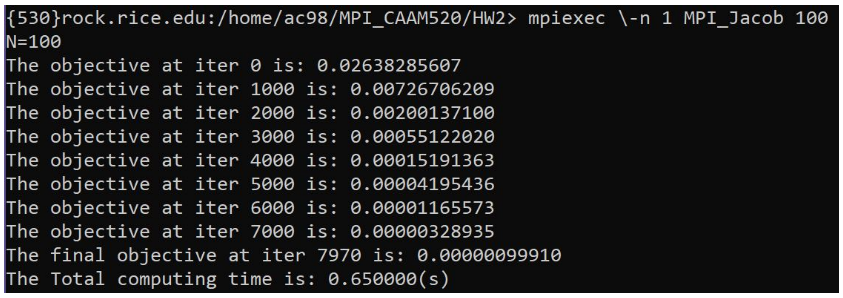
In this way, I am able to run MPI smoothly and improve the computing speed of the linear solver.

For the debugging strategy, I use many check point in the while loop and if there is a dead lock, I will be able to know which MPI process is stucked. I will also print out some key values and processor id in the debugging steps.

3 Weighted Jacobi method: Numerical results

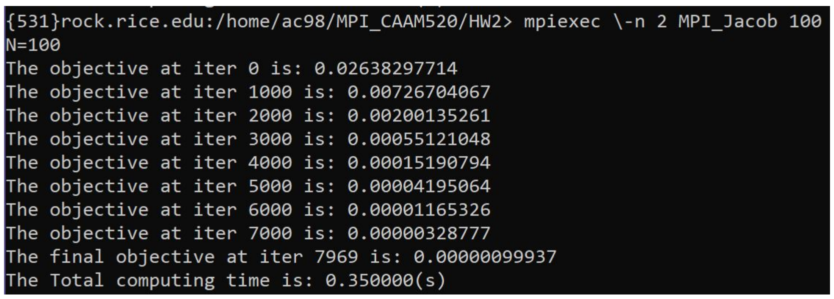
In this section, I will show the numerical result by runing the parallelized weighted Jacobi method. The objective function is the L2-norm of $b - Au$

For $N = 100$, the computing time by different number of processor is provided below.



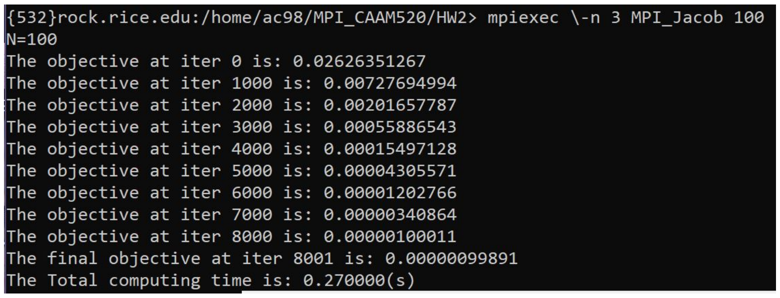
```
{530}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 1 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02638285607
The objective at iter 1000 is: 0.00726706209
The objective at iter 2000 is: 0.00200137100
The objective at iter 3000 is: 0.00055122020
The objective at iter 4000 is: 0.00015191363
The objective at iter 5000 is: 0.00004195436
The objective at iter 6000 is: 0.00001165573
The objective at iter 7000 is: 0.00000328935
The final objective at iter 7970 is: 0.00000099910
The Total computing time is: 0.650000(s)
```

Figure 2: Weighted Jacobi method on N=100, Nproc=1 situation



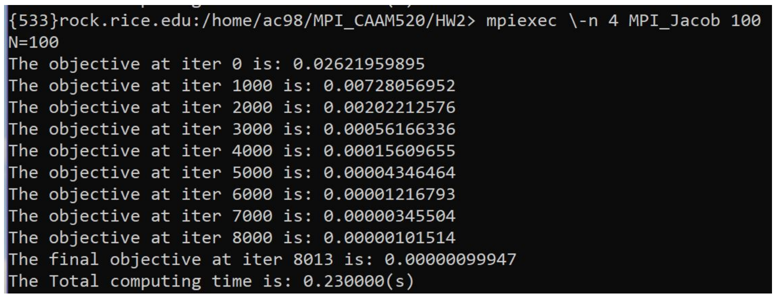
```
{531}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 2 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02638297714
The objective at iter 1000 is: 0.00726704067
The objective at iter 2000 is: 0.00200135261
The objective at iter 3000 is: 0.00055121048
The objective at iter 4000 is: 0.00015190794
The objective at iter 5000 is: 0.00004195064
The objective at iter 6000 is: 0.00001165326
The objective at iter 7000 is: 0.00000328777
The final objective at iter 7969 is: 0.00000099937
The Total computing time is: 0.350000(s)
```

Figure 3: Weighted Jacobi method on N=100, Nproc=2 situation



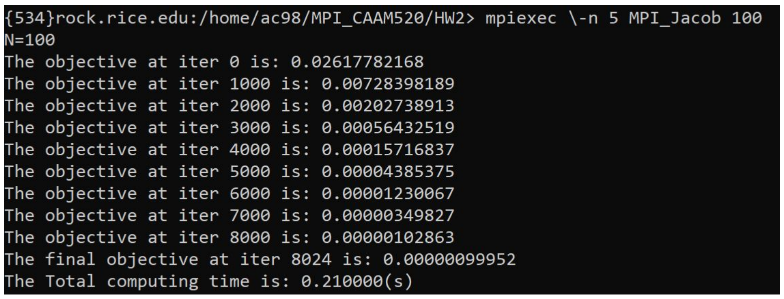
```
{532}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 3 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02626351267
The objective at iter 1000 is: 0.00727694994
The objective at iter 2000 is: 0.00201657787
The objective at iter 3000 is: 0.00055886543
The objective at iter 4000 is: 0.00015497128
The objective at iter 5000 is: 0.00004305571
The objective at iter 6000 is: 0.00001202766
The objective at iter 7000 is: 0.00000340864
The objective at iter 8000 is: 0.00000100011
The final objective at iter 8001 is: 0.00000099891
The Total computing time is: 0.270000(s)
```

Figure 4: Weighted Jacobi method on N=100, Nproc=3 situation



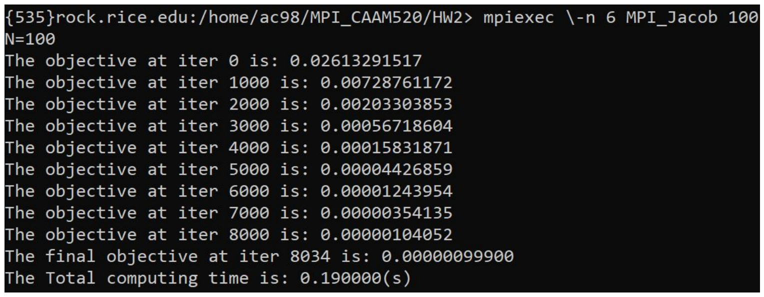
```
{533}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 4 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02621959895
The objective at iter 1000 is: 0.00728056952
The objective at iter 2000 is: 0.00202212576
The objective at iter 3000 is: 0.00056166336
The objective at iter 4000 is: 0.00015609655
The objective at iter 5000 is: 0.00004346464
The objective at iter 6000 is: 0.00001216793
The objective at iter 7000 is: 0.00000345504
The objective at iter 8000 is: 0.00000101514
The final objective at iter 8013 is: 0.00000099947
The Total computing time is: 0.230000(s)
```

Figure 5: Weighted Jacobi method on N=100, Nproc=4 situation



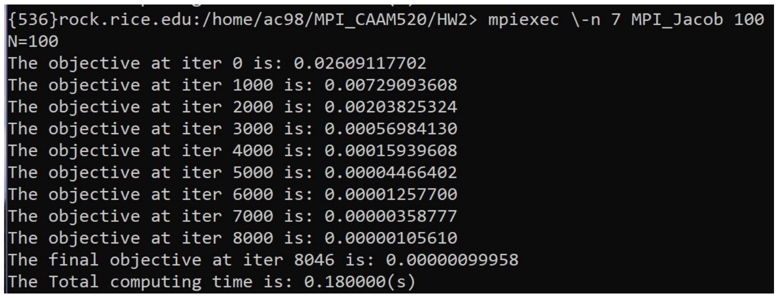
```
{534}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 5 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02617782168
The objective at iter 1000 is: 0.00728398189
The objective at iter 2000 is: 0.00202738913
The objective at iter 3000 is: 0.00056432519
The objective at iter 4000 is: 0.00015716837
The objective at iter 5000 is: 0.00004385375
The objective at iter 6000 is: 0.00001230067
The objective at iter 7000 is: 0.00000349827
The objective at iter 8000 is: 0.00000102863
The final objective at iter 8024 is: 0.00000099952
The Total computing time is: 0.210000(s)
```

Figure 6: Weighted Jacobi method on N=100, Nproc=5 situation



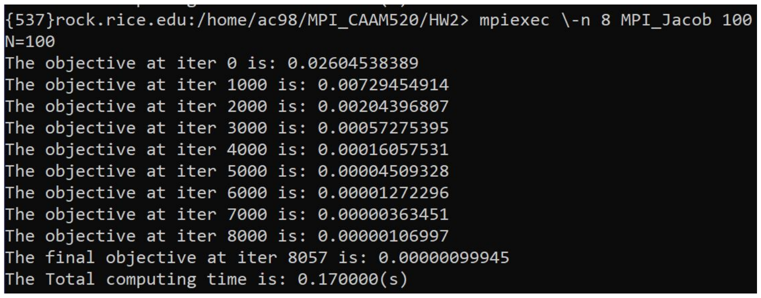
```
{535}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 6 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02613291517
The objective at iter 1000 is: 0.00728761172
The objective at iter 2000 is: 0.00203303853
The objective at iter 3000 is: 0.00056718604
The objective at iter 4000 is: 0.00015831871
The objective at iter 5000 is: 0.00004426859
The objective at iter 6000 is: 0.00001243954
The objective at iter 7000 is: 0.00000354135
The objective at iter 8000 is: 0.00000104052
The final objective at iter 8034 is: 0.00000099900
The Total computing time is: 0.190000(s)
```

Figure 7: Weighted Jacobi method on N=100, Nproc=6 situation



```
{536}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 7 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02609117702
The objective at iter 1000 is: 0.00729093608
The objective at iter 2000 is: 0.00203825324
The objective at iter 3000 is: 0.00056984130
The objective at iter 4000 is: 0.00015939608
The objective at iter 5000 is: 0.00004466402
The objective at iter 6000 is: 0.00001257700
The objective at iter 7000 is: 0.00000358777
The objective at iter 8000 is: 0.00000105610
The final objective at iter 8046 is: 0.00000099958
The Total computing time is: 0.180000(s)
```

Figure 8: Weighted Jacobi method on N=100, Nproc=7 situation



```
{537}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 8 MPI_Jacob 100
N=100
The objective at iter 0 is: 0.02604538389
The objective at iter 1000 is: 0.00729454914
The objective at iter 2000 is: 0.00204396807
The objective at iter 3000 is: 0.00057275395
The objective at iter 4000 is: 0.00016057531
The objective at iter 5000 is: 0.00004509328
The objective at iter 6000 is: 0.00001272296
The objective at iter 7000 is: 0.00000363451
The objective at iter 8000 is: 0.00000106997
The final objective at iter 8057 is: 0.00000099945
The Total computing time is: 0.170000(s)
```

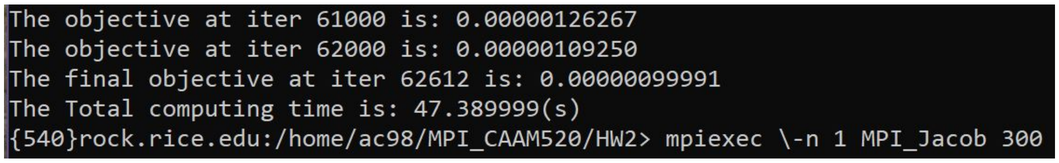
Figure 9: Weighted Jacobi method on N=100, Nproc=8 situation

Figure 2 to 9 shows the computing time for the weighted Jacobi method with different number of procs. The problem is of size 102×102 and the computing time for different parallelism is listed in the Table 1. The number of iterations for reaching the ideal misfit is approximately the same for different parallelism. For the computing time, when the number of processors is small, the improvements of computing time is obivious that I can obtain approximately $t = \frac{t_0}{Nproc}$ computing time by parallelism. However, for large N values ($N >= 4$), there is a lot of time wasted on the communication between processors. The improvments by using large number of processors are not obvious.

N processors	Computing time (s)
1	0.650
2	0.350
3	0.270
4	0.230
5	0.210
6	0.190
7	0.180
8	0.170

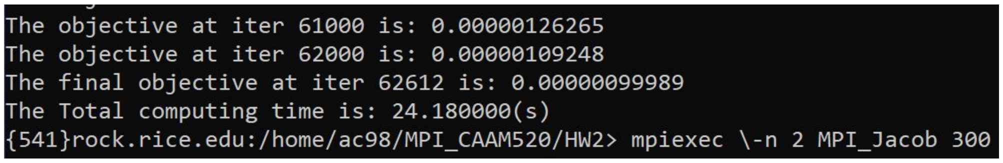
Table 1: Table for computing time using Parallelized Weighted Jacobi method N=100

For $N = 300$, the computing time by different number of processor is provided below.



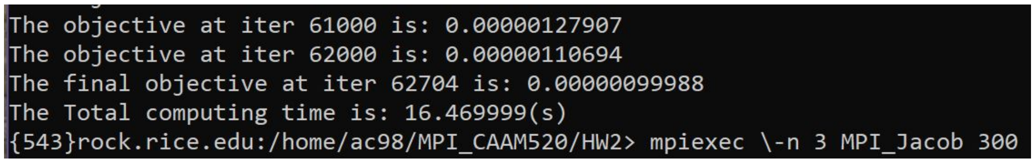
```
The objective at iter 61000 is: 0.00000126267
The objective at iter 62000 is: 0.00000109250
The final objective at iter 62612 is: 0.00000099991
The Total computing time is: 47.389999(s)
{540}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 1 MPI_Jacob 300
```

Figure 10: Weighted Jacobi method on N=300, Nproc=1 situation



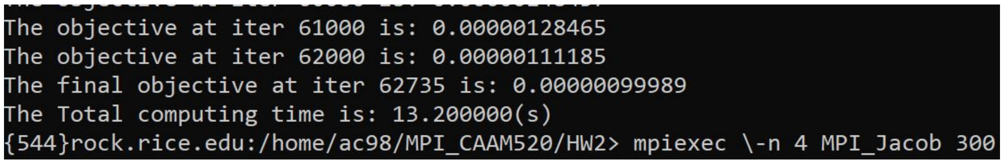
```
The objective at iter 61000 is: 0.00000126265
The objective at iter 62000 is: 0.00000109248
The final objective at iter 62612 is: 0.00000099989
The Total computing time is: 24.180000(s)
{541}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 2 MPI_Jacob 300
```

Figure 11: Weighted Jacobi method on N=300, Nproc=2 situation



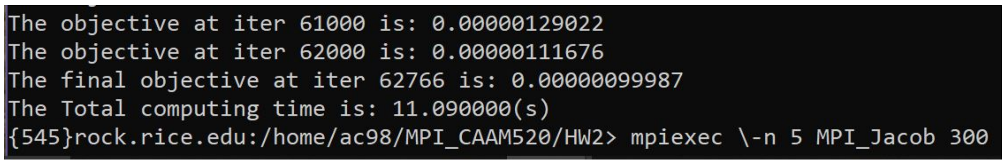
```
The objective at iter 61000 is: 0.00000127907
The objective at iter 62000 is: 0.00000110694
The final objective at iter 62704 is: 0.00000099988
The Total computing time is: 16.469999(s)
{543}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 3 MPI_Jacob 300
```

Figure 12: Weighted Jacobi method on N=300, Nproc=3 situation



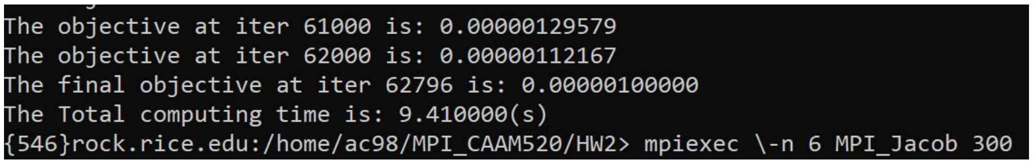
```
The objective at iter 61000 is: 0.00000128465
The objective at iter 62000 is: 0.00000111185
The final objective at iter 62735 is: 0.00000099989
The Total computing time is: 13.200000(s)
{544}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 4 MPI_Jacob 300
```

Figure 13: Weighted Jacobi method on N=300, Nproc=4 situation



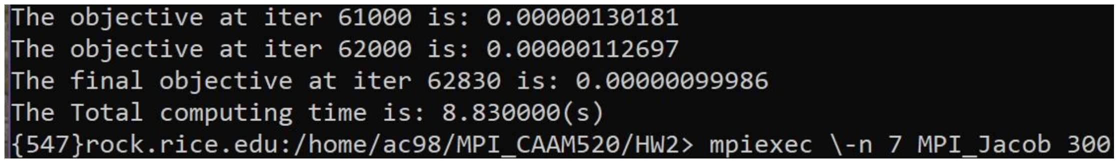
```
The objective at iter 61000 is: 0.00000129022
The objective at iter 62000 is: 0.00000111676
The final objective at iter 62766 is: 0.00000099987
The Total computing time is: 11.090000(s)
{545}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 5 MPI_Jacob 300
```

Figure 14: Weighted Jacobi method on N=300, Nproc=5 situation



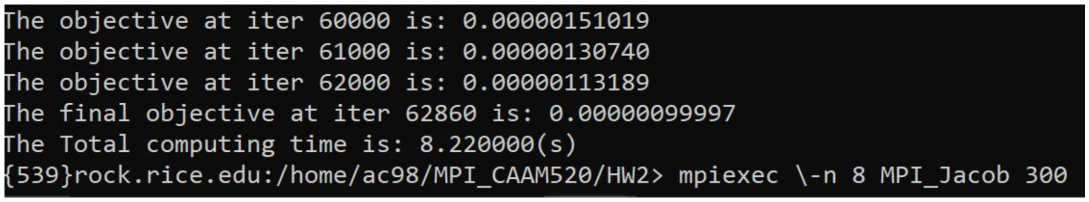
```
The objective at iter 61000 is: 0.00000129579
The objective at iter 62000 is: 0.00000112167
The final objective at iter 62796 is: 0.00000100000
The Total computing time is: 9.410000(s)
{546}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 6 MPI_Jacob 300
```

Figure 15: Weighted Jacobi method on N=300, Nproc=6 situation



```
The objective at iter 61000 is: 0.00000130181
The objective at iter 62000 is: 0.00000112697
The final objective at iter 62830 is: 0.00000099986
The Total computing time is: 8.830000(s)
{547}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 7 MPI_Jacob 300
```

Figure 16: Weighted Jacobi method on N=300, Nproc=7 situation



```
The objective at iter 60000 is: 0.00000151019
The objective at iter 61000 is: 0.00000130740
The objective at iter 62000 is: 0.00000113189
The final objective at iter 62860 is: 0.00000099997
The Total computing time is: 8.220000(s)
{539}rock.rice.edu:/home/ac98/MPI_CAAM520/HW2> mpiexec \-n 8 MPI_Jacob 300
```

Figure 17: Weighted Jacobi method on N=300, Nproc=8 situation

Figure 10 to 17 shows the computing time for the weighted Jacobi method with different number of procs. The problem is of size 302×302 and the computing time for different parallelism is listed in the Table 2. The number of iterations for reaching the ideal misfit is approximately the same for different parallelism. The new problem is approximately 9 times bigger than the first problem solving 102×102 linear equations. For the computing time, when the number of processors is small ($N <= 4$), the improvements of computing time is obivious. For instance, by using 2 processors I got the computing one half of the computing time using 1 processor. However, the performance is limited when we move to problem with large number of processors ($n > 4$). The computing time is similar between using 6, 7, and 8 processors.

N processors	Computing time (s)
1	47.390
2	24.180
3	16.470
4	13.200
5	11.090
6	9.410
7	8.830
8	8.220

Table 2: Table for computing time using Parallelized Weighted Jacobi method N=300

In sum, using parallelism helps improving the performance of linear solver, such as weighted Jacobi method, but for large number of processors, the computing time performance is limited due to the cost in communicating between processors. The parallelism needs to be optimized for large number of parallel processors.