PROJECT REPORT and READ_ME

I have implemented two-dimensional convolution in a jarvis cluster using different parallelization techniques and models. During the process I have practically learnt the advanced MPI techniques.

2-D convolution: Lets assume Im1 and Im2 are two N*N images, where each element is a complex number. The real part of the complex number will be non zero, while the complex part will be zero. The process starts by taking 2-D Fast Fourier Transformation (FFT) on each of the images, and then perform an matrix multiplication on the intermediate results obtained, followed by an inverse of the result.

A = 2D-FFT(Im 1) (task 1)
B = 2D-FFT(Im 2) (task 2)
C = MM_Point(A,B) (task 3)
D = Inverse_2DFFT(C) (task 4)

I have shared the screenshots of the execution of all the programs.

a) MPI (Send and Receive)

Compilation Command - mpicc -c MPI_sendrecv1.c
Compilation Command - mpicc -o MPI_sendrecv1 MPI_sendrecv1.o -lm
Running Command - mpirun -n 1 ./MPI_sendrecv1 (For 1 processor). Similarly change the number after the -n flag for the program to be run on any number of processors

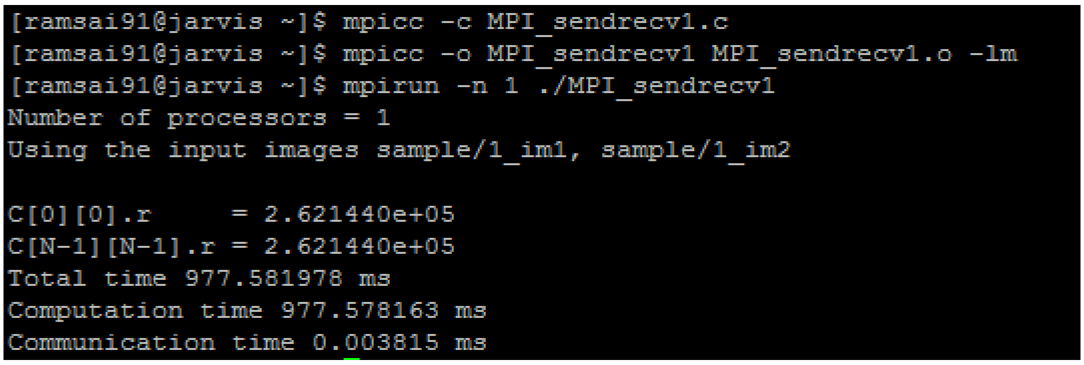
I have assumed that, the sequential time will be same as the time for code to be executed on a single processor. I have used MPI Send and Receive calls.

Speedup = (Time taken for serial computation/ Time taken for parallel computation)

Processors	Total Time (ms)	Computation Time (ms)	Communication Time(ms)	Speedup
P = 1	977	977	0	1
P = 2	922	532	390	1.83
P = 4	1783	286	1497	3.41
P = 8	3009	61	2948	16.01

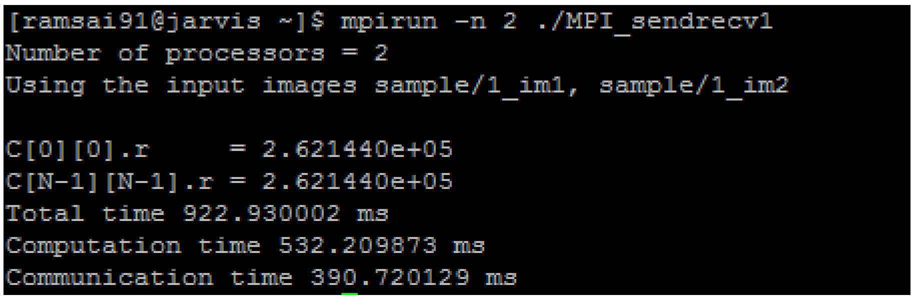
As the number of processors are increasing, the computation time is decreasing and the communication time between the processors are increasing.

1 Processor

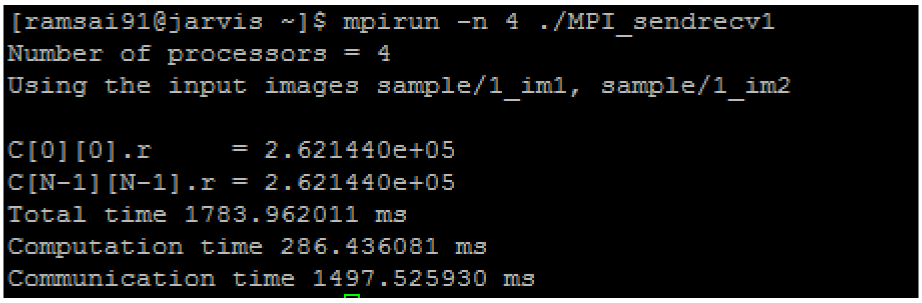


```
[ramsai91@jarvis ~]$ mpicc -c MPI_sendrecv1.c
[ramsai91@jarvis ~]$ mpicc -o MPI_sendrecv1 MPI_sendrecv1.o -lm
[ramsai91@jarvis ~]$ mpirun -n 1 ./MPI_sendrecv1
Number of processors = 1
Using the input images sample/1_im1, sample/1_im2

C[0][0].r     = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
Total time 977.581978 ms
Computation time 977.578163 ms
Communication time 0.003815 ms
```

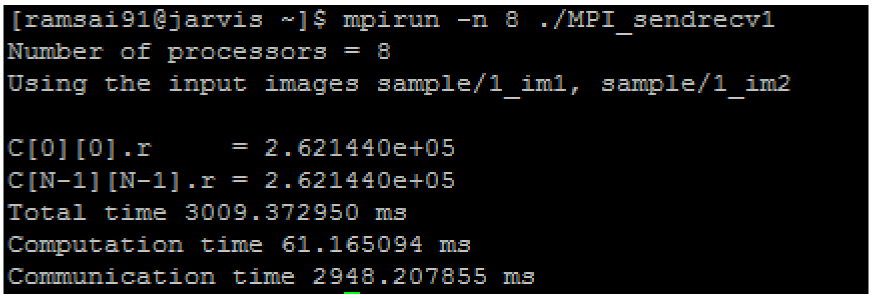
2 Processor


```
[ramsai91@jarvis ~]$ mpirun -n 2 ./MPI_sendrecv1
Number of processors = 2
Using the input images sample/1_im1, sample/1_im2

C[0][0].r      = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
Total time 922.930002 ms
Computation time 532.209873 ms
Communication time 390.720129 ms
```

4 Processor


```
[ramsai91@jarvis ~]$ mpirun -n 4 ./MPI_sendrecv1
Number of processors = 4
Using the input images sample/1_im1, sample/1_im2

C[0][0].r      = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
Total time 1783.962011 ms
Computation time 286.436081 ms
Communication time 1497.525930 ms
```

8 Processor


```
[ramsai91@jarvis ~]$ mpirun -n 8 ./MPI_sendrecv1
Number of processors = 8
Using the input images sample/1_im1, sample/1_im2

C[0][0].r      = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
Total time 3009.372950 ms
Computation time 61.165094 ms
Communication time 2948.207855 ms
```

b) MPI (scatter - gather)

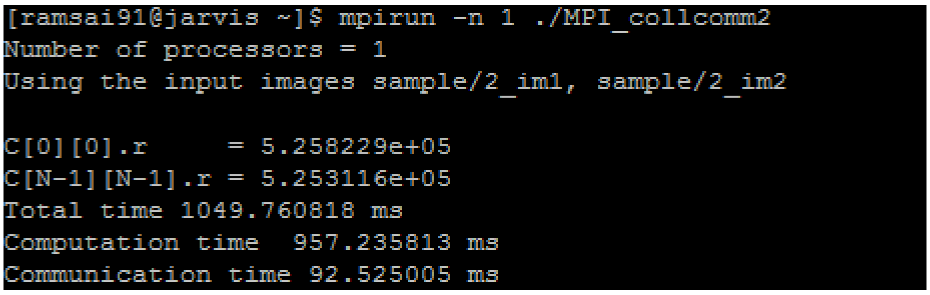
Compilation Command - mpicc -c MPI_collcomm2.c
Compilation Command - mpicc -o MPI_collcomm2 MPI_collcomm2.o -lm
Running Command - mpirun -n 1 ./MPI_collcomm2 (For 1 processor). Similarly change the number after the -n flag for the program to be run on any number of processors

I have assumed that, the sequential time will be same as the time for code to be executed on a single processor. I have used MPI scatter and gather calls.

Processors	Total Time (ms)	Computation Time (ms)	Communication Time(ms)	Speedup
P = 1	1049	957	92	1
P = 2	1295	534	760	1.79
P = 4	2376	241	2135	3.97
P = 8	3237	72	3165	13.29

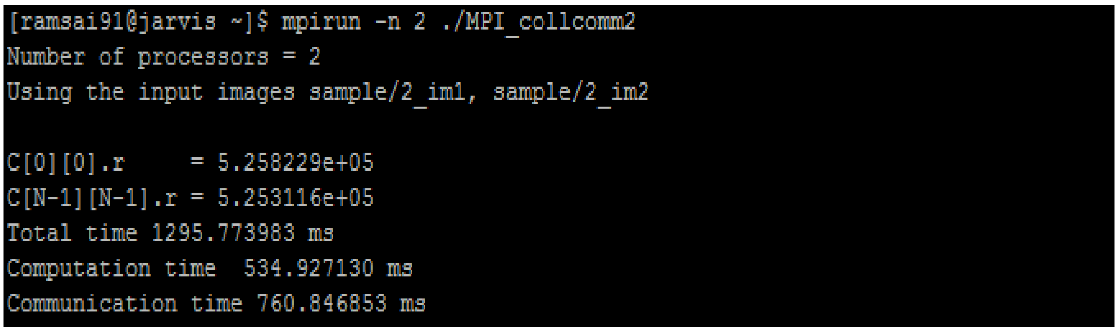
As the number of processors are increasing, the computation time is decreasing and the communication time between the processors are increasing.

1 Processor



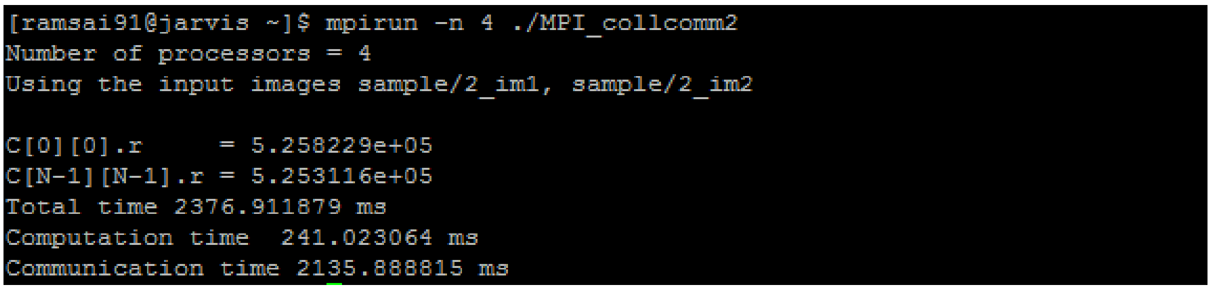
```
[ramsai91@jarvis ~]$ mpirun -n 1 ./MPI_collcomm2
Number of processors = 1
Using the input images sample/2_im1, sample/2_im2

C[0][0].r      = 5.258229e+05
C[N-1][N-1].r = 5.253116e+05
Total time 1049.760818 ms
Computation time  957.235813 ms
Communication time 92.525005 ms
```

2 Processor



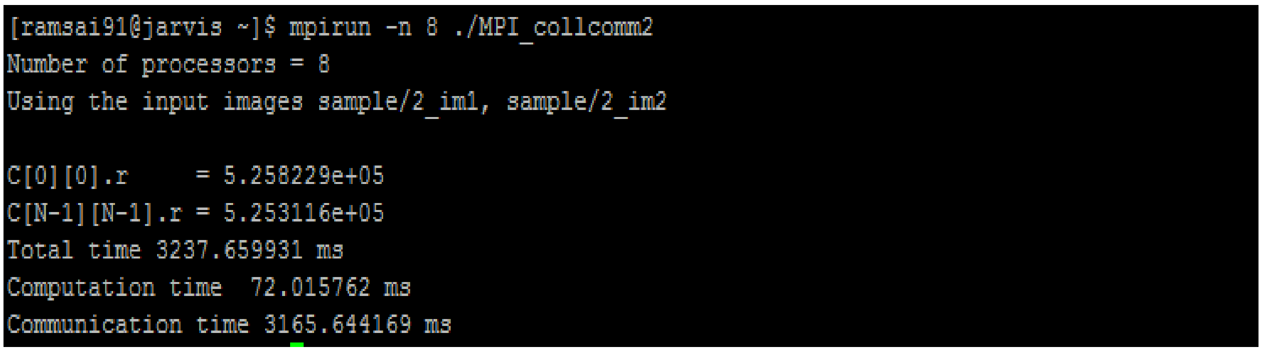
```
[ramsai91@jarvis ~]$ mpirun -n 2 ./MPI_collcomm2
Number of processors = 2
Using the input images sample/2_im1, sample/2_im2

C[0][0].r      = 5.258229e+05
C[N-1][N-1].r = 5.253116e+05
Total time 1295.773983 ms
Computation time  534.927130 ms
Communication time 760.846853 ms
```

4 Processor



```
[ramsai91@jarvis ~]$ mpirun -n 4 ./MPI_collcomm2
Number of processors = 4
Using the input images sample/2_im1, sample/2_im2

C[0][0].r      = 5.258229e+05
C[N-1][N-1].r = 5.253116e+05
Total time 2376.911879 ms
Computation time  241.023064 ms
Communication time 2135.888815 ms
```

8 Processor



```
[ramsai91@jarvis ~]$ mpirun -n 8 ./MPI_collcomm2
Number of processors = 8
Using the input images sample/2_im1, sample/2_im2

C[0][0].r      = 5.258229e+05
C[N-1][N-1].r = 5.253116e+05
Total time 3237.659931 ms
Computation time  72.015762 ms
Communication time 3165.644169 ms
```

c) Hybrid Programming (MPI + OpenMP)

Compilation Command - mpicc -c HybridProgramming.c
Compilation Command - mpicc -o HybridProgramming HybridProgramming.o -lm

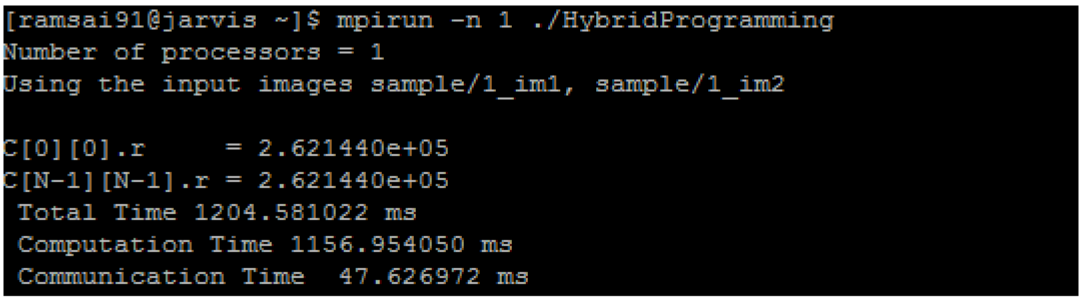
Running Command - mpirun -n 1 ./HybridProgramming (For 1 processor). Similarly change the number after the -n flag for the program to be run on any number of processors

I have assumed that, the sequential time will be same as the time for code to be executed on a single processor. I have used MPI calls combined with OpenMP for parallelization.

Processors	Total Time (ms)	Computation Time (ms)	Communication Time(ms)	Speedup
P = 1	1204	1156	47	1
P = 2	3407	1004	2402	1.15
P = 4	4042	1167	2874	0.99
P = 8	5627	1491	4136	0.77

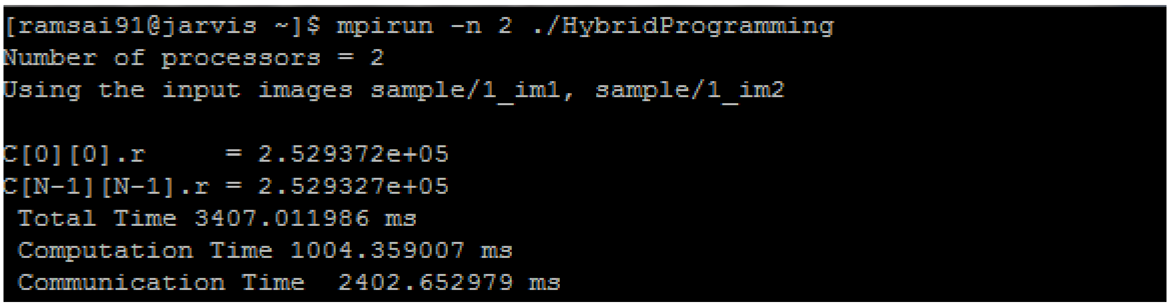
For Higher number of processors (4 or 8), the speedup is decreasing as the computation time takes more than the serial algorithm.

1 Processor



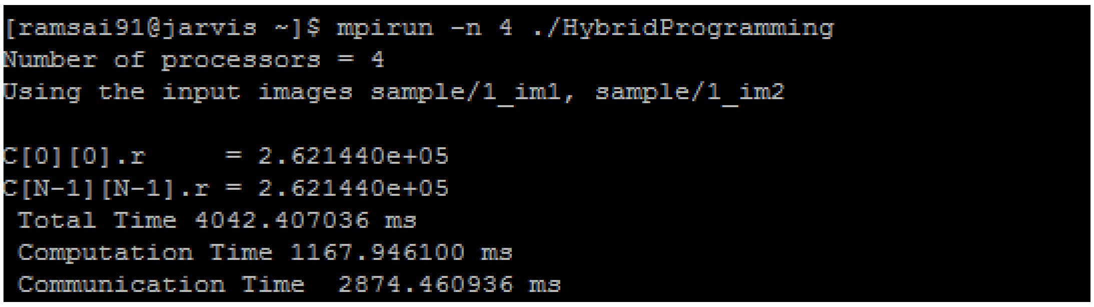
```
[ramsai91@jarvis ~]$ mpirun -n 1 ./HybridProgramming
Number of processors = 1
Using the input images sample/1_im1, sample/1_im2

C[0][0].r     = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
 Total Time 1204.581022 ms
 Computation Time 1156.954050 ms
 Communication Time  47.626972 ms
```

2 Processor



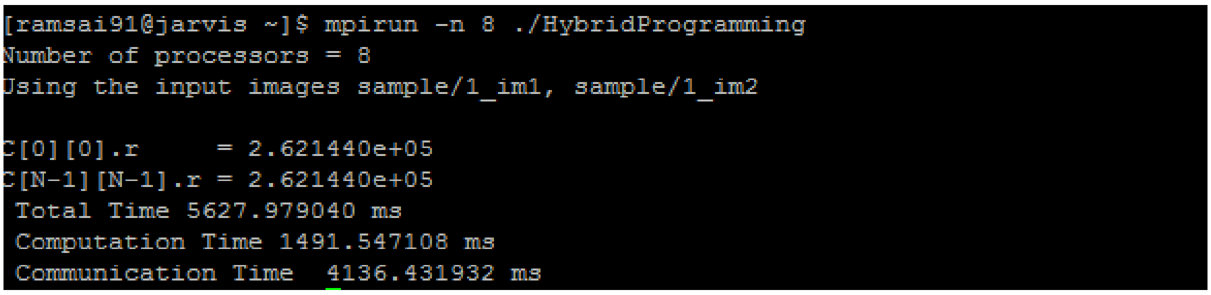
```
[ramsai91@jarvis ~]$ mpirun -n 2 ./HybridProgramming
Number of processors = 2
Using the input images sample/1_im1, sample/1_im2

C[0][0].r     = 2.529372e+05
C[N-1][N-1].r = 2.529327e+05
 Total Time 3407.011986 ms
 Computation Time 1004.359007 ms
 Communication Time  2402.652979 ms
```

4 Processor



```
[ramsai91@jarvis ~]$ mpirun -n 4 ./HybridProgramming
Number of processors = 4
Using the input images sample/1_im1, sample/1_im2

C[0][0].r     = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
 Total Time 4042.407036 ms
 Computation Time 1167.946100 ms
 Communication Time  2874.460936 ms
```

8 Processor



```
[ramsai91@jarvis ~]$ mpirun -n 8 ./HybridProgramming
Number of processors = 8
Using the input images sample/1_im1, sample/1_im2

C[0][0].r      = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
 Total Time 5627.979040 ms
 Computation Time 1491.547108 ms
 Communication Time  4136.431932 ms
```

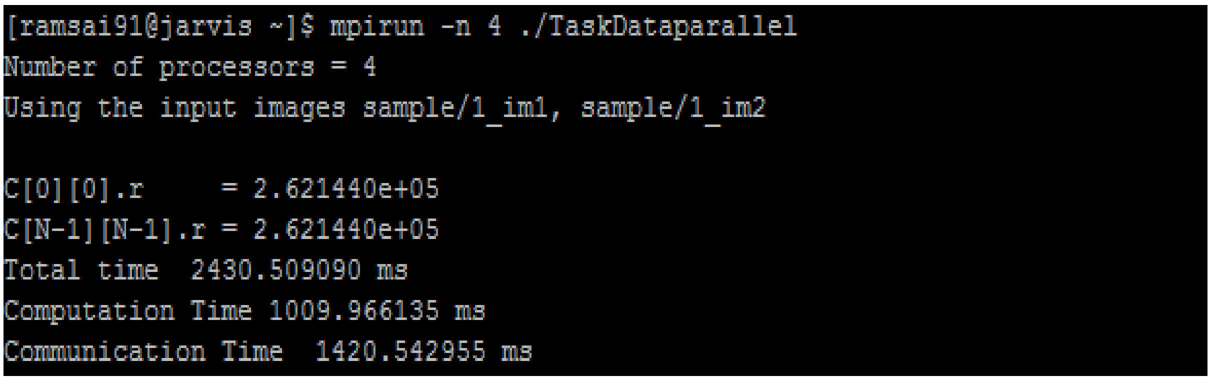
d) Task and Data parallel Programming

Compilation Command - mpicc -c TaskDataparallel.c
Compilation Command - mpicc -o TaskDataparallel TaskDataparallel -lm
Running Command - mpirun -n 4 ./MPI_TaskDataparallel (For 4 processor). Similarly change the number after the -n flag for the program to be run on any number of processors. (The number should be multiple of 4).

Since, For the given program, it takes four processors to find solution to the problem. I have assumed that the time taken to find solution for 4 processor as a base for calculating speedup. Give the processor size in multiple of 4, since the program has four different tasks to be done.

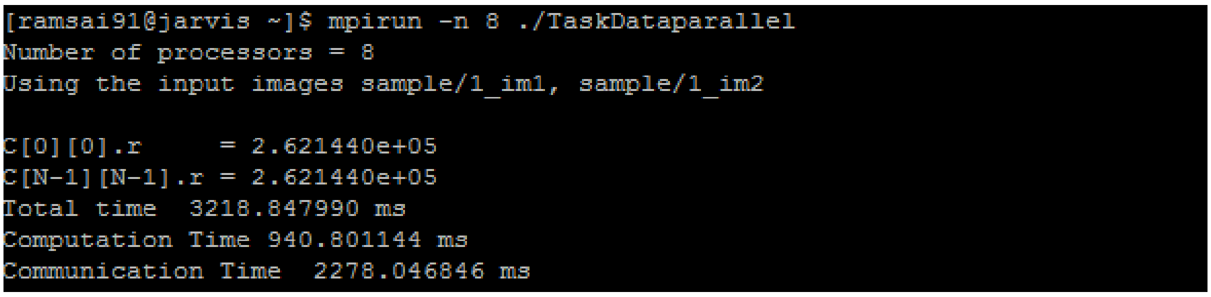
Processors	Total time (ms)	Computation time (ms)	Communication time (ms)	Speedup
P = 4	2430	1009	1420	1
P = 8	3218	940	2278	1.07

4 Processor



```
[ramsai91@jarvis ~]$ mpirun -n 4 ./TaskDataparallel
Number of processors = 4
Using the input images sample/1_im1, sample/1_im2

C[0][0].r      = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
Total time  2430.509090 ms
Computation Time 1009.966135 ms
Communication Time  1420.542955 ms
```

8 Processor



```
[ramsai91@jarvis ~]$ mpirun -n 8 ./TaskDataparallel
Number of processors = 8
Using the input images sample/1_im1, sample/1_im2

C[0][0].r      = 2.621440e+05
C[N-1][N-1].r = 2.621440e+05
Total time  3218.847990 ms
Computation Time 940.801144 ms
Communication Time  2278.046846 ms
```

As the number of processors are increasing, the computation time is decreasing and the communication time between the processors are increasing.

e) In case of 8 processors, The (A) MPI- Send and Receive has better performance. The reason is that, all the processors are working on some data all the time while some of the processors will be free in case of (D) Task and Data Parallel programming. When there are different tasks to be done, different processors will be working on different jobs. In such a case (D) Task and Data parallel programming will be much effective.