CS546 “Parallel and Distributed Processing”

Term Project

Suyog Kharage A20402686

Parallelization strategy:

A=2D-FFT(Im1)	(task1)
B=2D-FFT(Im2)	(task2)
C=MM_Point(A,B)	(task3)
D=Inverse-2DFFT(C)	(task4)

The parallel algorithm using MPI is as follows (root process = 0):

Step 1 - Root process loads input files
Step 2 - Root process scatters the inputs (A and B from task 1 and task 2 respectively) (Communication).
Step 3 - Perform 1D-FFT on rows of both input (computation)
Step 4 - Root process gathers above processed inputs (Communication)
Step 5 –Transpose the input in sequential manner (serial computation)
Step 6 - Root process scatters above processed inputs (Communication)
Step 7 - Perform 1D-FFT on columns of above processed inputs (Computation)
Step 8 - C is calculated by performing MM_Point(A,B). (Computation)
Step 9 - Perform Inverse 1D-FFT over rows of C (Computation)
Step 10 - Gather C at root process (Communication)
Step 11 - Perform transpose on C in sequential manner (serial computation).
Step 12 - Perform scatter operation on C from root process (Communication)
Step 13 - Perform Inverse 1D-FFT over rows of C (Computation)
Step 14 - Gather C at root process (Communication)
Step 15 - Write the output by root process

Step1 to Step 7 are performed under task 1 and task 2 (A and B)
Step 8 is performed under task 3 (C)
Step 9 to Step 15 are performed under task 4 (D)

The size of the matrices is taken as 512*512.

```
/* Size of matrix (NxN) */
const int N = 512;
```

A) MPI using Send and Recv functions

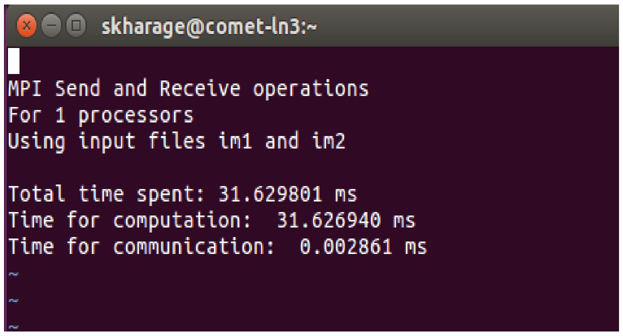
In this program, MPI_Send() and MPI_Recv() functions are used to perform send and receive operations.

The root process (0) is present in all communications acting as a master process.

MPI function *MPI_Wtime()* is used at the end of every step to calculate computation and communication times.

Tried the execution on 8 processors for multiple times. I observed that it is unstable because of communication time is higher. Below are the computation and communication times for processors 1,2,4 and 8.

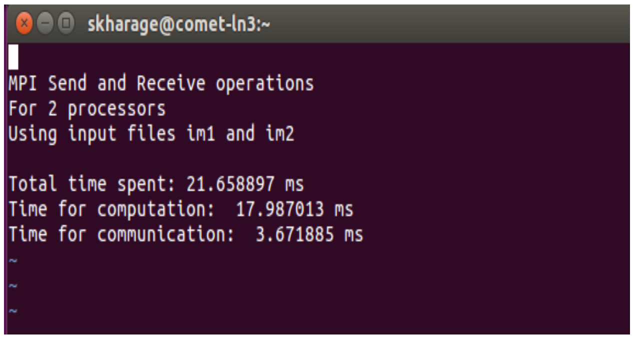
For 1 processor –



```
MPI Send and Receive operations
For 1 processors
Using input files im1 and im2

Total time spent: 31.629801 ms
Time for computation:  31.626940 ms
Time for communication:  0.002861 ms
~
~
~
```

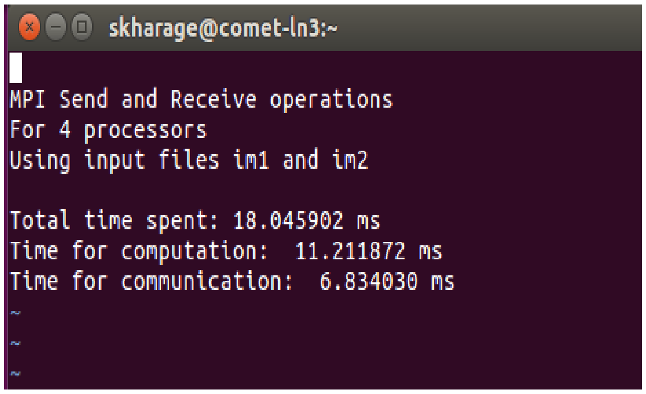
For 2 processors –



```
MPI Send and Receive operations
For 2 processors
Using input files im1 and im2

Total time spent: 21.658897 ms
Time for computation:  17.987013 ms
Time for communication:  3.671885 ms
~
~
~
```

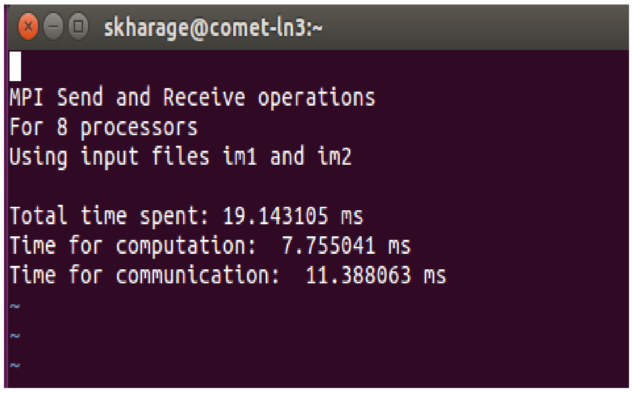
For 4 processors –



```
MPI Send and Receive operations
For 4 processors
Using input files im1 and im2

Total time spent: 18.045902 ms
Time for computation:  11.211872 ms
Time for communication:  6.834030 ms
~
~
~
```

For 8 processors –



```
MPI Send and Receive operations
For 8 processors
Using input files im1 and im2

Total time spent: 19.143105 ms
Time for computation:  7.755041 ms
Time for communication:  11.388063 ms
~
~
~
```

As we can see, time for 8 processors is increased as compared to other number of processors. This is due to more communication time between those 8 processors.

Speedup –

For 2 processors – 1.46

For 4 processors – 1.75

For 8 processors – 1.65

B) MPI using collective calls

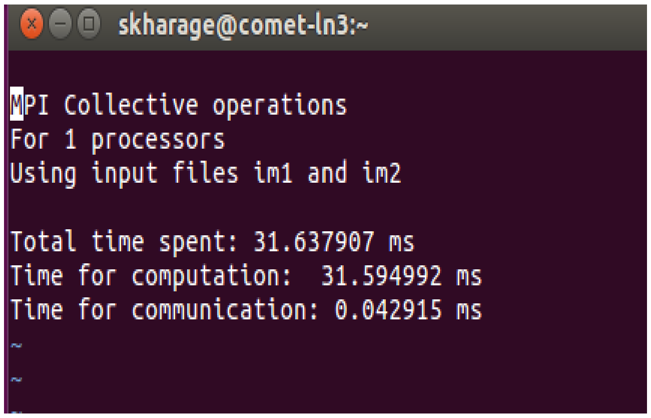
For this task, MPI_Scatter() and MPI_Gather() functions are used.

There is not so much difference in results.

From below results, we can see that, communication time is improved sometimes.

We can conclude that the collective calls improve communication performance, but this is not true for every time.

For 1 processor –

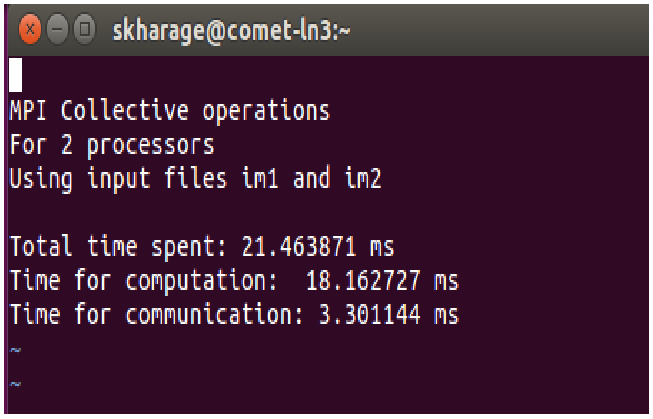


```
skharage@comet-ln3:~

MPI Collective operations
For 1 processors
Using input files im1 and im2

Total time spent: 31.637907 ms
Time for computation:  31.594992 ms
Time for communication: 0.042915 ms
~
~
```

For 2 processors-

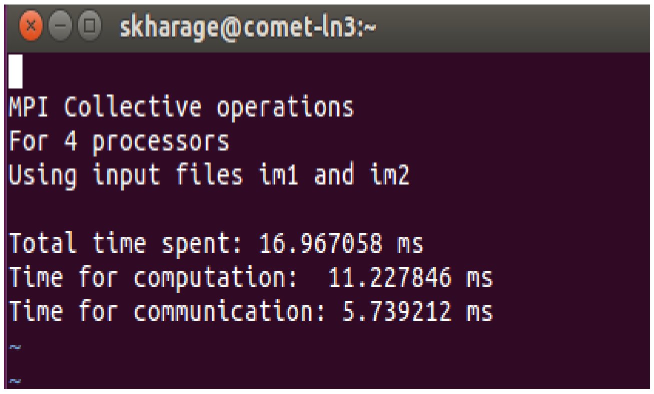


```
skharage@comet-ln3:~

MPI Collective operations
For 2 processors
Using input files im1 and im2

Total time spent: 21.463871 ms
Time for computation:  18.162727 ms
Time for communication: 3.301144 ms
~
~
```

For 4 processors –

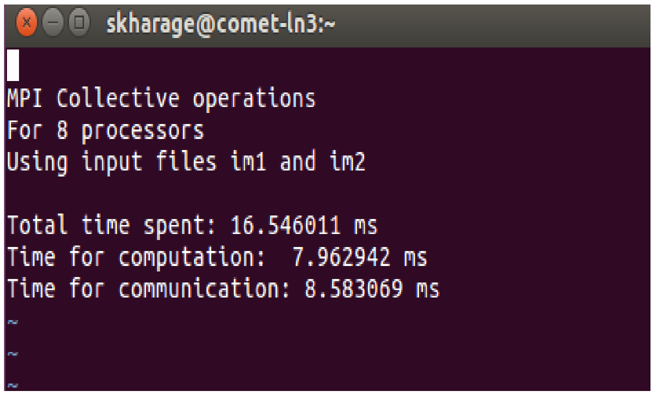


```
skharage@comet-ln3:~

MPI Collective operations
For 4 processors
Using input files im1 and im2

Total time spent: 16.967058 ms
Time for computation:  11.227846 ms
Time for communication: 5.739212 ms
~
~
```

For 8 processors –



```
skharage@comet-ln3:~

MPI Collective operations
For 8 processors
Using input files im1 and im2

Total time spent: 16.546011 ms
Time for computation:  7.962942 ms
Time for communication: 8.583069 ms
~
~
~
```

Speedup-

For 2 processors – 1.47
For 4 processors – 1.86
For 8 processors – 1.91

C) Task parallelism

In this program, all processing units are divided into 4 groups, and each one of them performs one task. We are using 8 processors hence; each group will have 2 processors. If there are 16 processors, then there could be 4 groups with 4 processors in each group.

Each processor group has been grouped using *MPI_Group*, and internal communications are performed using specific communicators, called *P1_comm*, *P2_comm*, *P3_comm* and *P4_comm.*

For 8 processors:

- P1 group will have P1_comm communicator with processors of rank 0 and 1.
- P2 group will have P2_comm communicator with processors of rank 2 and 3.

- P3 group will have P3_comm communicator with processors of rank 4 and 5.
- P4 group will have P4_comm communicator with processors of rank 6 and 7.

Root processor reads and distributes input. Root processor loads input and scatters A and B to all processors of group P1 (which comes under communication).

Task 1-
Group P1 performs 2D-FFT on part A:
Step 1 – Group P1 performs 1D-FFT on part A (computation).
Step 2 – First process in P1 gathers A from other P1 processes (communication).
Step 3 - First process in P1 perform transpose on A (serial computation).
Step 4 - First process in P1 perform scatter operation on A to other processes in P1 (communication).
Step 5 – All processes in P1 perform 1D-FFt on A (computation)
Step 6 – Each process of P1 group sends its chunk to corresponding P3 process. (communication)

Task 2-
Group P2 performs 2D-FFT on part B in parallel to task 1:
Step 1 – Group P2 performs 1D-FFT on part B (computation).
Step 2 – First process in P2 gathers A from other P2 processes (communication).
Step 3 - First process in P2 perform transpose on B (serial computation).
Step 4 - First process in P2 perform scatter operation on B to other processes in P2 (communication).
Step 5 – All processes in P2 perform 1D-FFt on B (computation)
Step 6 – Each process of P2 group sends its chunk to corresponding P3 process. (communication)

Task 3 -

In this task, point to point multiplication is performed by P3 group after completion of tasks 1 and 2:

Step 1 – Processes in P3 perform point to point multiplication of A and B to get C (Computation).

Step 2 – Every P3 process sends its chunk of C to corresponding P4 process (communication).

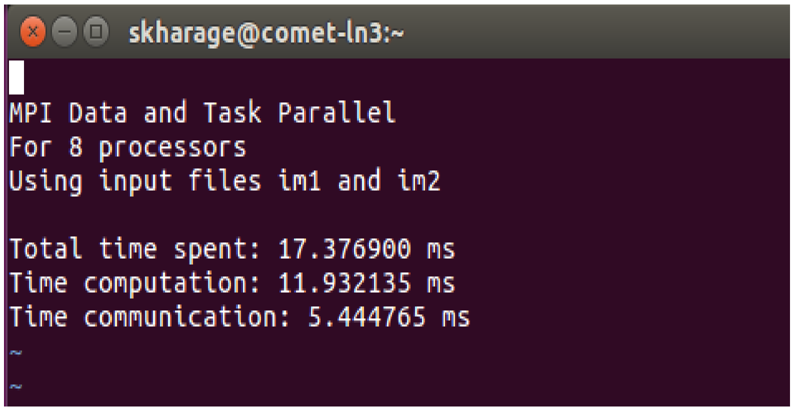
Task 4-
Group P4 performs inverse 2D-FFT on C, after task 3:

Step 1 – Group P4 performs inverse 1d-FFT on C (Computation).
Step 2 - First process of P4 gathers C from other P4 processes (Communication).
Step 3 - First process of P4 perform transpose operation on C (serial computation).
Step 4 - First process of P4 scatters C to other P4 processes (Communication).
Step 5 – Group P4 perform inverse 1D-FFT on C (Computation).
Step 6 – Every P4 process sends its chunk of C to the root process (Communication).

Gather output (combined chunks of C) from all processes of P4 and write it to file.

This algorithm requires more communications because of two additional communications for task 3.

Execution time for 8 processors-


```
skharage@comet-ln3:~

MPI Data and Task Parallel
For 8 processors
Using input files im1 and im2

Total time spent: 17.376900 ms
Time computation: 11.932135 ms
Time communication: 5.444765 ms
~
~
```

Speedup:

For 8 processors – 1.82

D) Comparison between cases A and C

For 8 processors, the performance is much better in case A in terms of computation time and the performance is much better in case C in terms of communication time. The reason is that task and data is divided among processes which causes less communication. Hence, communication time is less in C.

Potential of the D will be more in case of multiple jobs, different groups could be working in different jobs.

Appendix: source code

The source code can be found together with this document.

- MPI_Send_Recv.c: version of the program that uses MPI with Send & Recv

- MPI_Collective.c: version of the program that uses MPI with collective calls

- MPI_Task_Data_Parallel.c: version of the program that uses task parallelism with four groups