Objective of program is to perform following tasks.

A=2D-FFT (Im1) (task1)

B=2D-FFT (Im2) (task2)

C=MM_Point (A, B) (task3)

D=Inverse-2DFFT(C) (task4)

Where A, B, C, and D are N*N arrays of complex. D is the final output.

Im1 and Im2 are two N*N images, where each element is a complex number.

Parallelization strategy

The above tasks are implemented in program using different parallelization strategy. The different parallelization techniques are implemented here are

a. Use MPI send and receive operations to perform communication.
b. Use MPI collective communication functions.
c. Use hybrid programming like MPI and OpenMP.
d. Use a Task and Data Parallel Model.

a. Use MPI send and receive operations to perform communication.

1. Source processor sends matrices A and B to all other processors.
2. Each processor will get matrices size divided by number of processor
3. FFT is applied on matrices A and B
4. Source processor receives matrices A and B from all other processors
5. Matrices A and B are transposed.
6. Source processor scatters A and B to all other processor
7. FFT is applies to matrices A and B
8. Matrix C is calculated by point to pint multiplication of matrices A and B.

9. Inverse FFT is applied on matrix C.
10. Source processor receives matrix C from all other processor.

Performance results:

No of processors	Total time (s)	Computation Time (s)	Communication Time (s)	Speed-up
P=1	1.24	1.24	0.00	
P=2	1.43	0.47	0.96	1.15
P=4	1.99	0.32	1.67	1.6
P=8	2.34	0.097	2.25	1.88

b. Use MPI collective communication functions.

MPI collective communication functions are scatter and gather.

1. Source processor scatters matrices A and B to all other processors.
2. Each processor will get matrices size divided by number of processor
3. FFT is applied on matrices A and B
4. Source processor gathers matrices A and B from all other processors
5. Matrices A and B are transposed.
6. Source processor scatters A and B to all other processor
7. FFT is applies to matrices A and B
8. Matrix C is calculated by point to pint multiplication of matrices A and B.
9. Inverse FFT is applied on matrix C.
10. Source processor gathers matrix C from all other processor.

No of processors	Total time (s)	Computation Time (s)	Computation Time (s)	Speed-up
P=1	1.07	1.06	0.01	
P=2	2.04	0.45	1.59	1.89
P=4	2.93	0.22	2.17	2.22
P=8	3.63	0.07	3.56	3.36

c. Use hybrid programming like MPI and openMP.
   1. Source processor scatters matrices A and B to all other processors.
   2. Each processor will get matrices size divided by number of processor
   3. FFT is applied on matrices A and B using openMP parallel directive using 8 threads.
   4. Source processor gathers matrices A and B from all other processors
   5. Matrices A and B are transposed using openMP parallel directive using 8 threads.
   6. Source processor scatters A and B to all other processor
   7. FFT is applies to matrices A and B using openMP parallel directive using 8 threads.
   8. Matrix C is calculated by point to pint multiplication of matrices A and B using openMP parallel directive using 8 threads.
   9. Inverse FFT is applied on matrix C using openMP parallel directive using 8 threads.
   10. Source processor gathers matrix C from all other processor.

Threads = 8

No of processors	Total time (s)	Computation Time (s)	Computation Time (s)	Speed-up
P=1	1.13	1.13	0.00	
P=2	2.95	1.13	1.82	2.61
P=4	3.46	1.02	2.45	3.06
P=8	5.10	1.45	3.61	4.51

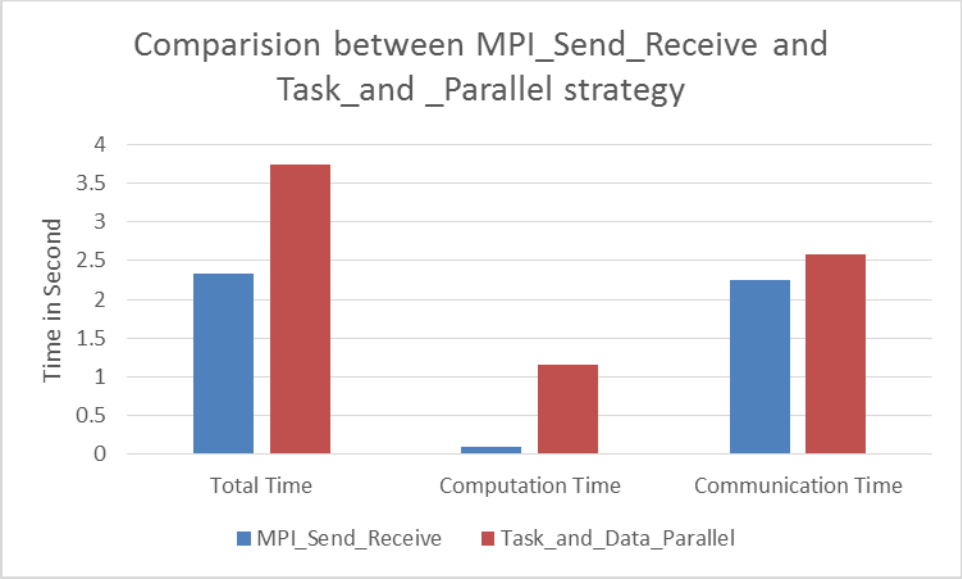
d. Use a Task and Data Parallel Model.

1. Processors are divided into four groups.
2. Create processor communicator for each group.
3. Source processor group sends matrices A and B to all other processors.
4. Apply FFT on matrix A, B in all processor groups
5. Receive matrices A and B from all processor groups.
6. Transpose matrices A and B
7. Send transposed matrices A, B to processor group one and two.
8. Apply FFT on matrices on A and B
9. Receive matrices A B to processor group three.
10. Calculate matrix C by point to point multiplication of matrices A and B.
11. Send matrix C to processor group three and four.
12. Apply Inverse FFT on matrix C.
13. Receive matrix C from processor group three and four.
14. Transpose matrix C.
15. Send transpose matrix C to processor group.
16. Apply Inverse FFT on matrix C.
17. Receive matrix C from processor group.

Execution time for P1=P2=P3=P4=2x

No of processors	Total time (s)	Computation Time (s)	Computation Time (s)	Speed-up
Px=2	3.74	1.15	2.59	1.41

Comparing the results obtained in (a) - (d) experiments for the case of 8 processors.



The performance obtained in experiment (a) using MPI send receive functions are better compared to experiment (d) using task and data parallel model.

Since (a) MPI send and Receive is implemented using SPMD model .i.e. Single processor and multiple data, accessing stack memory is very fast so communication cost and communication cost is less.
(b) Task and data parallel model, here each task process has its own stack memory so each task take more time to communicate with other task process, so communication and computation task is more expensive.

From this we can conclude that (a) has better performance compared to (d).

APPENDIX: SOURCE CODE

SOURCE FILES: I have place source code files for each parallel model in Jarvis at /home/vbidari/Term_Project/programs. Source code files are as follows

a. MPI_Send_Recieve.c
b. MPI_Collective_Communication.c
c. MPIopenMP.c
d. Task_and_Data_Parallel_Model.c

COMPILE & RUN: Refer readme.docx for complete compilation and running each program. I have placed in working directory at /home/vbidari/Term_Project.

Bash files for running above each source are placed in Jarvis at /home/vbidari/Term_Project/programs. Bash files are in same order for each source mentioned above.

a. MPI_Send_Recieve.bash
b. MPI_Collective_Communication.bash
c. MPI_OpenMP.bash
d. Task_Data_Parallel.bash

OUTPUT FILES: Output matrix for above each parallel model program are generated in same folder as above in Jarvis folder. Following are output matrix files for each above program. Generated output matrix files are as follows

a. output_matrix_MPI_Send_Recieve
b. output_matrix_MPI_Collective
c. output_matrix_MPI_OpenMP
d. output_matrix_Task_and_Data_Parallel