# CS546
# Final Project Report

# Name: Shreyas Bidwai
# CWID: A20403315


## Description

This project entails implementing two dimensional convolutions on parallel computers using different parallelization techniques and models. The objectives are to design and implement parallel programs using different models for parallelization and communication. 2-D convolution can be implemented by first taking 2-D Fast Fourier Transform of each input image, then perform point-wise multiplication of the intermediate results from the 2D FFTs, followed by an inverse 2-D FFT.

The parallelization strategy that is being used in this project is that the task is divided into sub tasks as given in the project description.

A=2D-FFT(im1) (Task 1)
B=2D-FFT(im2) (Task 2)
C=MM_Point(A,B) (Task 3)
D=inverse-2DFFT(C) (Task 4)

The parallel algorithm using MPI is as follows:-

1. First, the source process loads the input.
2. Then the source process distributes the tasks 1 and 2
3. 1D- FFT is performed on the rows of A and B
4. Then the source process gather the computations of A and B
5. And Transpose them sequentially
6. Then the source process again distributes A and B and computes the 1D-FFT over rows(which were cols) in A and B
7. And then the value of C is calculated by MM_point(A,B)
8. And then inverse 1D-FFT is performed over rows in C

9. The source program collects the value of C
10. And transpose of C is performed sequentially
11. Then source process distributes the value of C
12. Inverse 1D-FFT is performed over rows in C
13. Source program gather C
14. And writes the output to the output file.

While developing 3 versions of this algorithm, the input file im1 and im2 were used. The output was written to another file of the same format.

## A. MPI send and receive operations:

The algorithm used is same as explained above. The process (0) acts as the source process. MPI_Wtime() function is used to calculate the time taken for computations. The run times for different number of processors are:

P=1 the time is 0.034026 sec.
P=2 the time is 0.028053 sec
P=4 the time is 0.035612 sec
P=8 the time is 0.038085 sec

The processor communication time increases as the number of processors increases because, the processors need to communicate and send and receive data from a lot of processors, therefore the time required for communication is high.

## B. MPI collective communication calls

MPI_Bcast function is used. Broadcast is one of the standard collective communication techniques. During broadcast, one process sends all the data to all the processors in the communicator. Here, I have used MPI broadcast with MPI send and receive functions. Here, the main process sends the data

to all the processes and all the other processes receive the data from the main process. The run times for different number of processors are:

P=1 the time is 0.037185 sec.
P=2 the time is 0.028523 sec
P=4 the time is 0.036740 sec
P=8 the time is 0.034629 sec

The result doesn't change much. The communication time although is improved in collective communication calls when the number of processors is increased, because collective communication calls provide better response while inter process communication.

## C. MPI using Task Parallelism

The processing units are divided into 4 groups and each one of them will be charge of one task. Here, we are using 8 processors, therefore each will have 2. Each group of processors has been grouped with MPI_Group function.
Each group of processors has been grouped though MPI_Group and internal communication are made using specific intra-communicators, called P1_comm, P2_comm, P3_comm, and P4_comm.
The group is divided as P1 with 0, 1 ranks and P2 with 2, 3 ranks and P3 with 4, 5 ranks and P4 with 6, 7 ranks. The algorithm is little different. Here source process doesn't perform the serial function rather it is divided into groups. Task 1 is performed by P1, Task 2 is performed by P2, Task 3 is performed by P3, and Task 4 is performed by P4. The time taken for computation is 0.110106 sec.

## Comparison between results (a) – (c) for 8 processors

For 8 processors the performance is much better in (a) and (b) as compared to (c). The reason is that all processors are working all the time, while in

case of task parallelism there are always free processors at one particular moment.

In case of task parallelism, if the number of tasks increases to a very large extent, we can say that task parallelism performs better because different groups would be working on different jobs. And as such al processors would be busy.

## Appendix: Source Code

The source code for the 3 different programs is present in the folder as they were very large.
The names of the file are:

**MPI send and receive operation**: mpi_sr.c
**MPI collective communication:** mpi_comm.c
**MPI data and task parallelism:** mpi_task.c

The source code files are also present on comet@SDSC with same names in the folder project/.