CS546 "Parallel and Distributed Computing"
Term Project

Submission:

- ***Due by 5:00pm of 05/09/2013***
- ***NO extension is allowed.***
- ***You are expected to submit a report (maximum 5 pages) showing the results and your parallelization strategy and performance results. You are also expected to submit the source code of your programs as appendix to your report.***
- ***Your project will be graded based on the correctness and performance***
- ***Please upload your assignment on the Blackboard with the following name: CS546_SectionNumber_LastName_FirstName_Project. The submission should include:) the C source files, binary file, README.txt file, and report***

This project entails implementing two-dimensional convolution 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.

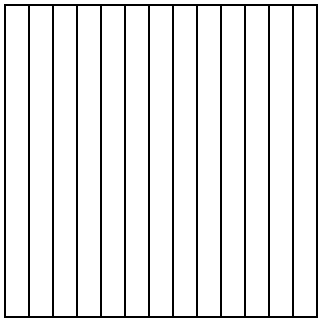
2-D convolution: assume Im1 and Im2 are two N*N images, where each element is a complex number (in reality an input image will have its real part as non-zero, while complex part will be zero). 2-D convolution can be implemented by first taking 2-D Fast Fourier Transform of each input image, then perform point-wise multiplication of the intermediate results from the 2D FFTs, followed by an inverse 2-D FFT. That is,

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. We are only interested in the real part of D.

2D-FFT: a 2D-FFT can be performed using 1D-FFT routines (which will be provided to you). 2D-FFT is obtained by first performing N, N-point 1D-FFTs along rows followed by N, N-point 1D-FFT along columns of the intermediate result of the row-fft step.

Row FFT



Column FFT

On a P-processor system, 2D FFT can be implemented by decomposing the data along rows, each processor performing local row FFTs on its part of data, then transposing the data, followed by another round of row FFTs (which are columns of the intermediate data because of the transpose).

This project requires you to do the following:

(a) Implement the 2D convolution using **SPMD model** and use **MPI send and receive operations** to perform communication. You need to run your program on 1, 2, 4, and 8 processors and provide speedups as well as computation and communication timings.
(b) Implement 2D convolution using **SPMD model** but use **MPI collective communication functions** wherever possible. You need to run your program on 1, 2, 4 and 8 processors and provide speedups as well as computation and communication timings.
(c) Implement 2D convolution model using **SPMD model** using **hybrid programming (MPI-openMP or MPI-Pthreads, Lecture 18).** You need to run your program on 1, 2, 4 and 8 processors, with 8 threads per processors. You need to provide speedups as well as computation and communication timings.
(d) Implement 2D convolution model using **a Task and Data Parallel Model**. You also need to show the use of communicators in MPI. Let's say we divide the P processors into four groups: P1, P2, P3, and P4. You will run Task 1 on P1 processors, Task 2 on P2 processors, Task 3 on P3 processors, and Task 4 on P4 processors. The Following figure illustrates this case. Report computation and communication results for P1=P2=P3=P4=2.
(e) Compare the results obtained in (a) - (d) for the case of 8 processors. Do not compute time for loading/writing data files.

Note:
All data files are 512*512 sizes. The data files contain 512*512 floating point values stored in ascii form with newline separating a row from the next(row 0 is written followed by \n then row 1 etc…). Your program must read these data files and generate an output file in the same format.

The 1D-FFT source code "fft.c" and two sets of sample input and output files (set #1: "1_im1","1_im2", and "out_1"; set #2: "2_im1", "2_im2", and "out_2") are provided on the blackboard.

The following is a simple C loop to read the input:

```
FILE *f; /*open file descriptor */
for (i=0;i<512;i++)
     for (j=0;j<512;j++)
          fscanf(fp,"%g",&data[i][j]);
```

This is a simple C loop to write the output:

```
for (i=0;i<512;i++) {
          for (j=0;j<512;j++)
           fprintf(fp,"%6.2g",&data[i][j]);
     fprintf(fp,"\n");
};
```