Homework 2: MPI

Distributed Systems M21

September 4, 2021

1 Introduction

We will be learning and implementing parallel computing solutions using the Message Passing Interface (MPI) (in C/C++) implemented in OpenMPI package.

2 Setup

You will be working in C/C++ using the OpenMPI library, and will be utilizing the server with the access provided in your mail.

With the given username and password you can connect to the server by:
ssh <username >@rce.iiit.ac.in,
and enter the password when prompted.

Strict actions will be taken against students with suspicious or not expected behaviour on the server. All actions are recorded. An 'F' grade will be given in case of any intentional antics.

Please use the Template provided in the Appendix (at the end). The library is already installed on the server, and you will need to load it before being able to work with the library.

The script given (check Appendix for reference), creates a job on the node, loads the library, compiles the given source code and proceeds to run it with the number of processes, input and output files. (You can use it as is, just change these relevant variables.

- Documentation: `https://www.rookiehpc.com/mpi/docs/index.php`
- References: `https://pages.tacc.utexas.edu/~eijkhout/pcse/html/index.html`, `http://condor.cc.ku.edu/~grobe/docs/intro-MPI-C.shtml`
- Tutorials: `https://mpitutorial.com/tutorials/`, `https://www.rookiehpc.`

`com/mpi/exercises/index.php`

3 Problems

Please refer the teams-list for the teammates and Question ID you will be working with.

There are 3 sets of questions: 1 and 2, 3 and 4, and 5 and 6. You have been assigned 3 questions, 1 each from these sets. The marks distribution for a question from each set is: 10, 5 and 10. (Possible problem set for example: questions 2, 4 and 5 of 10, 5 and 10 marks respectively).

Please note that doing any question not assigned to you will not be considered.

The input and output filenames will have to be passed as arguments to the program. The program should hence read the input data from the input file and save the output in the output file.

3.1 Question 1

Let A be a non-singular square matrix of n rows and n columns. **Find the inverse of A.** You can assume that A is partitioned into rows and the partitions are stored across the processors. You can also assume that for the given inputs an inverse always exists. **Use the row reduction method.**

Input: First line will contain N, the size of the square matrix. The next N rows will contain N values each, where the value in the i^{th} row at the j^{th} place is the A[i][j] value.

Output: A $N*N$ matrix which is the inverse of the given input matrix. That is N lines, each of N values, where the value in the i^{th} row and at j^{th} place denotes the $A^{-1}[i][j]$ value.

Constraints: $1 <= N <= 10^2$

Example:

Input 1:
3
7 2 1
0 3 -1
-3 4 -2

Output 1:
-2 8 -5
3 -11 7

9 -34 21

3.2 Question 2

Use the following recursive method to multiply two square matrices A and B of size $N * N$. (You can assume that N is a power of 2.) The recursive method divides A and B into four sub-matrices of size $\frac{N}{2} * \frac{N}{2}$.
Let A_{11}, A_{12}, A_{21},andA_{22} be the sub-matrices of A and B_{11}, B_{12}, B_{21},andB_{22} are the sub-matrices of B. We build C as four sub-matrices with

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}, C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$

and so on. Each of these multiplications are also done recursively by subdividing A_{11} etc. into four sub-matrices till a certain limit. The matrices A and B are partitioned by rows and stored across the processors.

Input: A single integer, N, which is the size of the square matrices A and B. The next N rows will contain N values each, where the value in the i^{th} row at the j^{th} place is the A[i][j] value. The next N rows will contain N values each, where the value in the i^{th} row at the j^{th} place is the B[i][j] value.

Output: N lines, each of N values, denoting the result, C, of multiplying A and B, where the value in the i^{th} row at the j^{th} place is the C[i][j] value.

Constraints: $1 <= N <= 10^2$

Example:

Input 1:

```
4
5 7 9 6
-2 3 -3 8
8 8 2 3
3 3 4 -8
3 1 6 -4
7 1 -4 9
-9 3 6 2
3 5 -4 7
```

Output 1:

```
1 69 32 103
66 32 -74 85
71 37 16 65
```

-30 -22 62 -33

3.3 Question 3

Estimate the value of sin(x) for an angle x using the definition that

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

.

Your program should take as input the precision to be achieved, i.e., for an accuracy of 6, the result should be the correct estimation to the 10^{-6} decimal place.

Input: A float value, X, the angle (in radians), and another integer value, P, to give the precision to be achieved.

Output: The result accurate to the given precision.

Example:

Input 1: 60 5

Output 1: 0.86603

3.4 Question 4

There is a numerical identity that the sum of the reciprocals of the squares of integers converges to $\frac{\pi^2}{6}$. Check this for a given N, where you will need to calculate $\frac{1}{1^2} + \frac{1}{2^2} + ... + \frac{1}{N^2}$.

Input: An integer, N.

Output: A float value denoting the result, closest to the 10^{-8} decimal place.

Constraints: $1 <= N <= 10^6$

Example:

Input 1: 10000

Output 1: 1.64483407

3.5 Question 5

Given an undirected graph G, find the number of triangles (i.e. the cycles of length 3) and the number of cycles of length 4 in the graph.

Input: An integer value, N, for the number of vertices, and another integer, E, for the number of edges. Following E lines, each with two integers (vertices), showing an edge exists between these two vertices.

Output: Two integers, the number of cycles of length 3, and the number of cycles of length 4.

Constraints: $2 <= N <= 10^2$, $1 <= E <= 10^3$

Example:

Input 1:
7 9
1 2
2 3
2 4
3 4
3 5
1 5
1 6
6 7
7 5
Output 1:
1 2

3.6 Question 6

A proper vertex coloring of a graph is such that, no adjacent vertices have the same color.

Given an undirected graph G, find a proper vertex coloring (some integer value) of the graph using Delta(G) + 1 colors or fewer. (where Delta(G) is the maximum degree of the graph) You may use the algorithm of Luby but aren't limited to it.

Input: An integer value, N, for the number of vertices, and another integer, E, for the number of edges. Following E lines, each with two integers (vertices), showing an edge exists between these two vertices.

Output: An integer, K, denoting the number of colors used. N integers, denoting the coloring of the i^{th} vertex.

Constraints:

Example:

Input 1:
5 6
1 2
1 4
1 3

2 5
3 4
4 5
Output 1:
3
1 2 2 3 1

4 Submission Details

The submission should be done in the form of a zip file (Naming convention: Team_<Team id>.tar.gz) .
The directory structure should be such:

- Directories (Naming convention: <Question id>/) for each question assigned to your team, with the respective source codes (Naming convention: <Question id>.cpp) in those .
- A README.md to briefly explain your code and algorithm.

Example structure

```
Team02
 ├─2
 │ └─2.cpp
 ├─3
 │ └─3.cpp
 ├─5
 │ └─5.cpp
 └─README.md
```

There will be a preliminary automated evaluation. Therefore, please strictly adhere to the expected format and handle appropriate edge cases, or you will be penalized.

The main purpose of this homework is to learn and implement parallel approaches to these problems, hence sequential or unsatisfactory parallel approaches will be penalized heavily. Plagiarism isn't tolerated (between students, or from the net), and if found strict actions will be taken.

5 Appendix: Template

```cpp
#include <iostream.h>
#include <mpi.h>
using namespace std;

int main( int argc, char **argv ) {
    int rank, numprocs;

    // initiate MPI
    MPI_Init( &argc, &argv );

    // get size of the current communicator
    MPI_Comm_size( MPI_COMM_WORLD, &numprocs );

    // get current process rank
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    /*synchronize all processes*/
    MPI_Barrier( MPI_COMM_WORLD );
    double start_time = MPI_Wtime();

    // enter your code here

    MPI_Barrier( MPI_COMM_WORLD );
    double end_time = MPI_Wtime() - start_time;
    double maxTime;
    // get max program run time for all processes
    MPI_Reduce( &end_time, &maxTime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD );
    if ( rank == 0 ) {
        cout<<"Total time (s): "<<maxTime<<"\n";
    }

    // shut down MPI and close
    MPI_Finalize();
    return 0;
}
```

6 Appendix: Script to run a MPI job

Please use this script to run your MPI jobs. Please leave the lines 3-6 as is, changes (unless advised) to them may cause issues and consequently access revocation. You have access to max 8 cores (the -n flag) In line 14, you will only need to change the number of processes (the -np flag, here 3) and the input and output file names.

```
1 #!/bin/bash
```

```

#SBATCH -N 2
#SBATCH -n 8
#SBATCH --mem-per-cpu=1024

#load the module on the node
module load hpcx-2.7.0/hpcx-ompi

#compile the program
mpic++ sum_recip.cpp

#run the executable
mpirun -np 3 -mca coll ^hcoll --use-hwthread-cpus ./a.
    out input.txt output.txt
```

7 Appendix: Relevant server commands

- sbatch run_mpi.sh : Command to run the batch job. (You will use it to execute your program.)
- squeue -u <username >: To get the status of your batch job.
- scancel <job id >: To cancel any batch job (Get the job id from squeue command)