Name: Yifeng Zhang

Date: Oct 26, 2019

Note: please read readme.txt to know the files detain in my submission folder

**CSS534 Program2**

**Parallelizing Wave Diffusion with MPI and OpenMP**

**1. Purpose**

In this assignment, parallelize a sequential version of a two-dimensional wave diffusion program using a hybrid form of MPI and OpenMP.

**2. Algorithms:**

Using Schroedinger's Wave Dissemination(formula) to compute the 2D square bucket.

**3. Parallelization strategies**

**3.1. MPI**

Set the Rank0 as master rank and other are slave/worker rank. Run the all the codes which cannot be parallelized in Rank0;

Allocate each rank a whole 3 \* size \* size matrix when initialize the z[3][size][size]. Calculate the whole z[0] at each rank.

From Time = 1, each rank only computing 1/mpi\_size(#rank) of whole computation for each step. Before starting the computation, each rank need exchange their boundary message and then start to compute the current time data.

To avoid deadlock in communication when exchange the boundary message, let odd ranks send first and then receive the message, while even ranks receive first and then send the message.

When printing, each work rank need send the partial of data which is only be computed in this rank to master rank(Rank0). After master rank (Rank0) receive all the message, print all the data in the master rank.

**3.2 OpenMP**

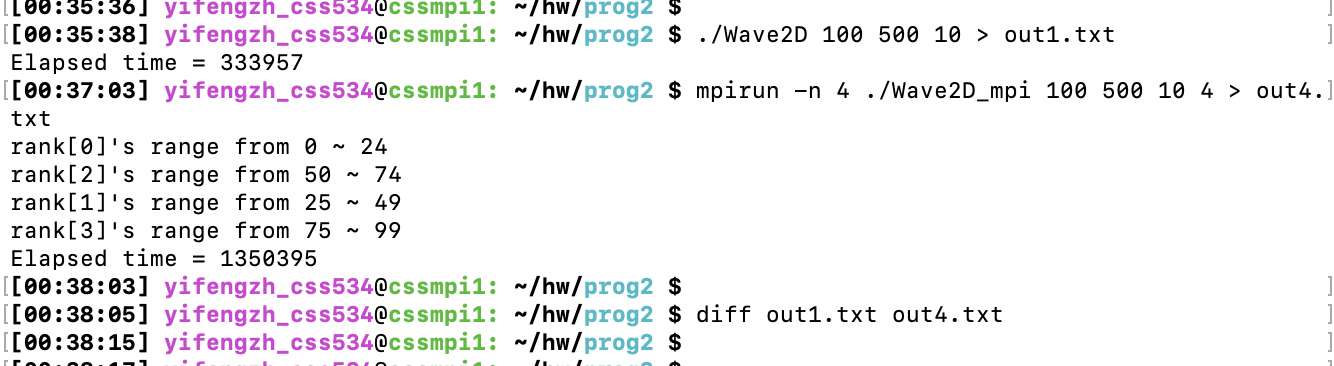
Set the #Threads in each rank (computer node) after start MPI. When each rank is computing, parallelize the calculation for loops (start from t = 1) with OpenMP.

**4. Result**

The result: I got multiple results and pick one of best result.

I completed the Wave2D.cpp and Wave2D\_mpi.cpp with MPI and OpenMP.

Step1: Compile the Wave2D.cpp and Wave2D\_mpi.cpp. Run the code with same parameters to verify the correctness of my Wave2D\_mpi.cpp and Wave2D.cpp. My code shows that no difference should be detected between a sequential and a parallel execution.



Step2: Run Wave2D\_mpi with the following scenarios:

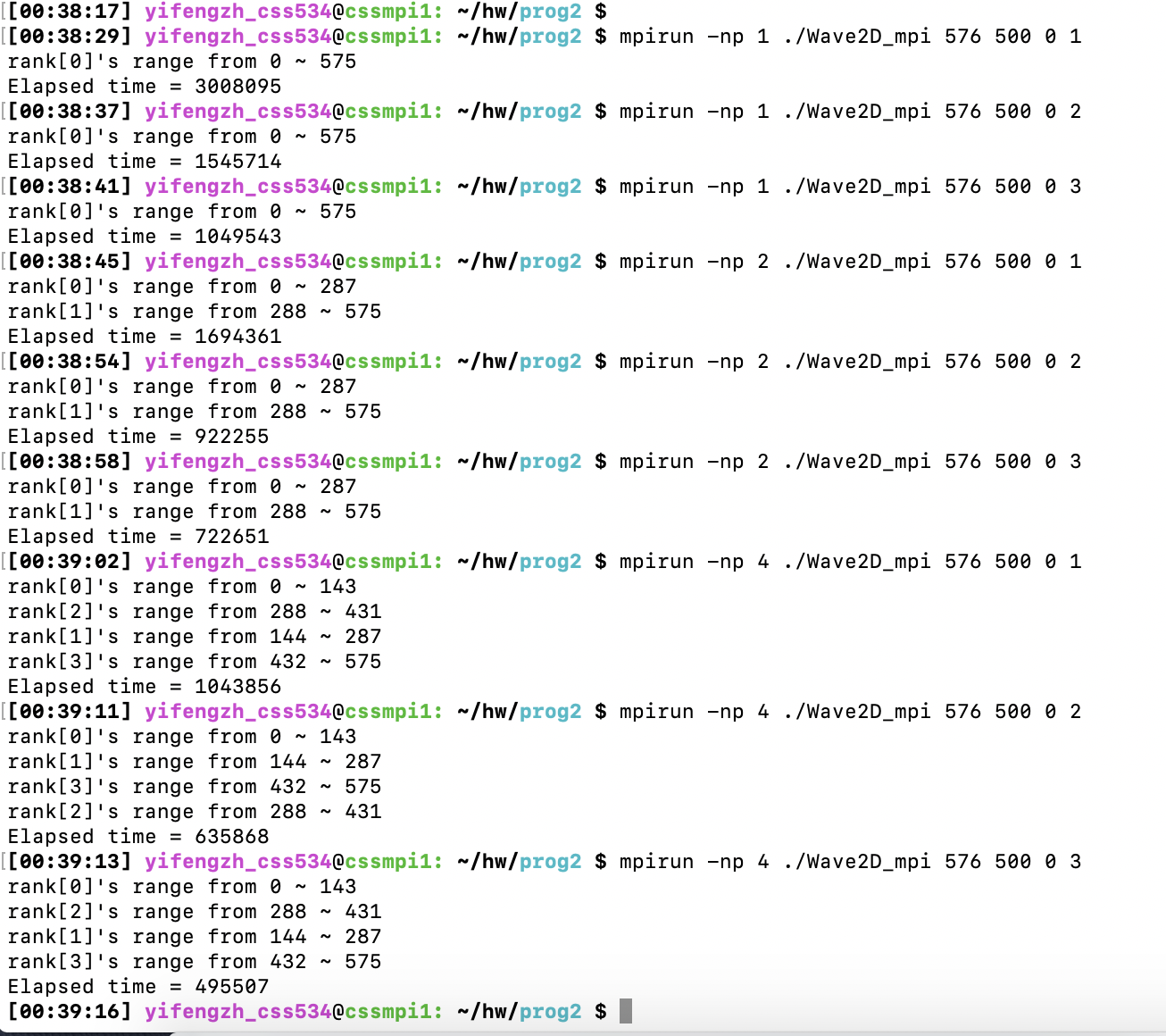
**mpirun –n x ./Wave2D\_mpi 576 500 0 y**

where X should be 1 through to 4 ranks and Y should be 1 through to 3 threads.

Compare the performance improvement with different ranks (MPI) and different threads(OpenMp).

|  |  |  |  |
| --- | --- | --- | --- |
| #Ranks /#Threads | 1 | 2 | 3 |
| 1 | 3008095/**3008095** = 1 | 3008095/**1545714** = **1.95** | 3008095/1049543 = **2.87** |
| 2 | 3008095/**1694361** = **1.77** | 3008095/**922255** = **3.26** | 3008095/722651 = **4.16** |
| 4 | 3008095/**1043856** = **2.88** | 3008095/**635868** = **4.73** | 3008095/495507 = **6.07** |

1. four MPI ranks perform 2.9 times better than sequential version.
2. four MPI ranks with 3 threads perform 6.1 times better than the sequential version.



5. **Discussions** about the parallelization, the limitation, and possible performance improvement

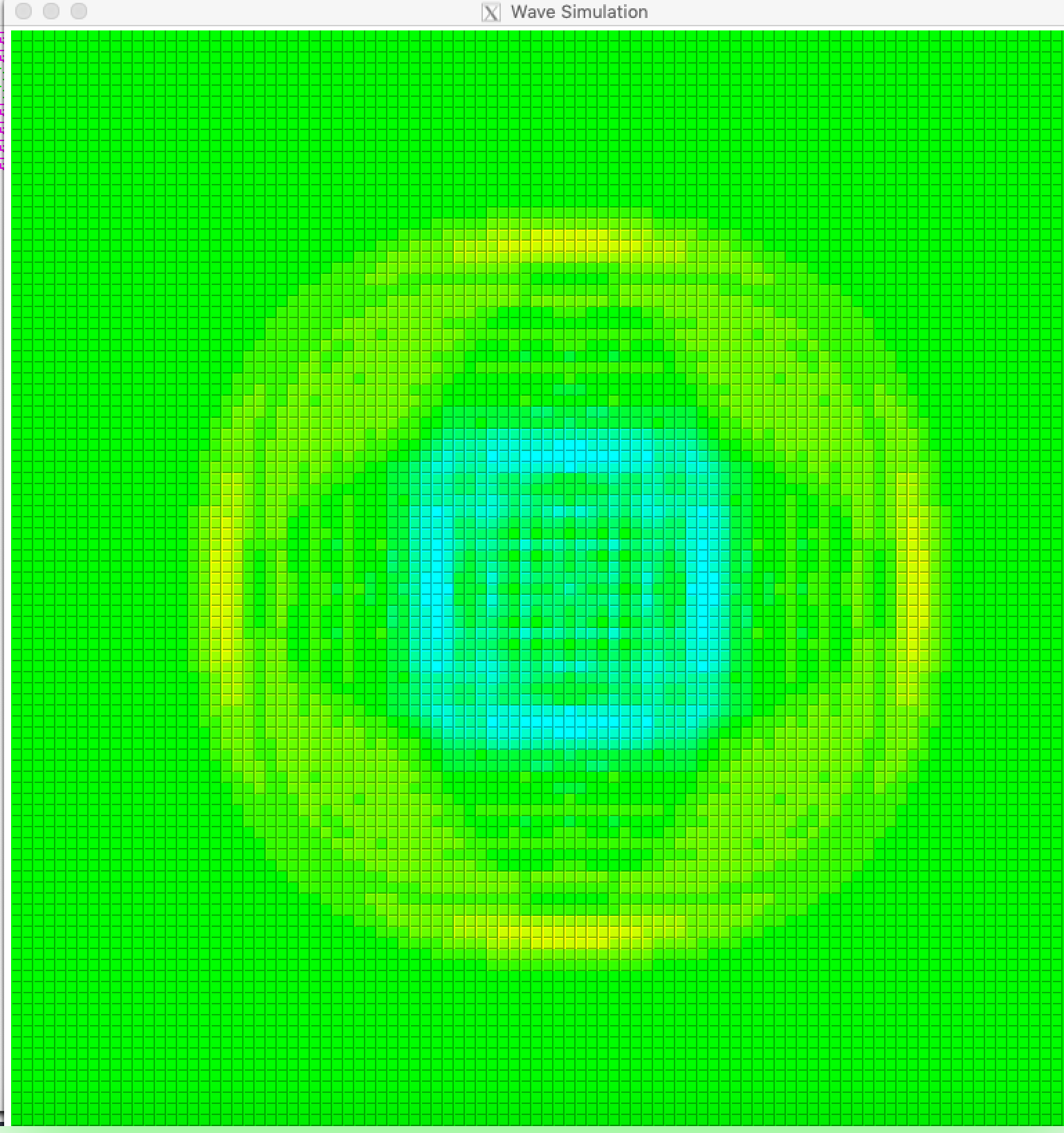
**Limitation:**

If need printing out the result, the performance is not getting improved. Data exchange spend lots time. It means when parallelization using MPI, if need collect all data together, it maybe uses more time to exchange data between ranks.

**Possible performance improvement:**

1. In rank0, using thread#1 to print out the available data in Rank0 and using thread#2 to receive data. After the data received from one rank (in order), check if threads#1 complete. if done, print the next available data.
2. Because send and receive message need more time, try to use 1 thread to send and receive message, and compute the boundary data(may plus less data), while other threads continue to compute the other non-boundary data.
3. For Schroedinger's Wave Dissemination algorithm, set the top, bottom, left, right boundary. Only Computing data from top – 1 to bottom + 1, left – 1 to right + 1.

6 . Other files: After running my Wave2D to get the data show in the below pictures.



Lab2 – Matrix multiplication using MPI.

Solving the Matrix multiplication using MPI. It’s a practice to use MPI.

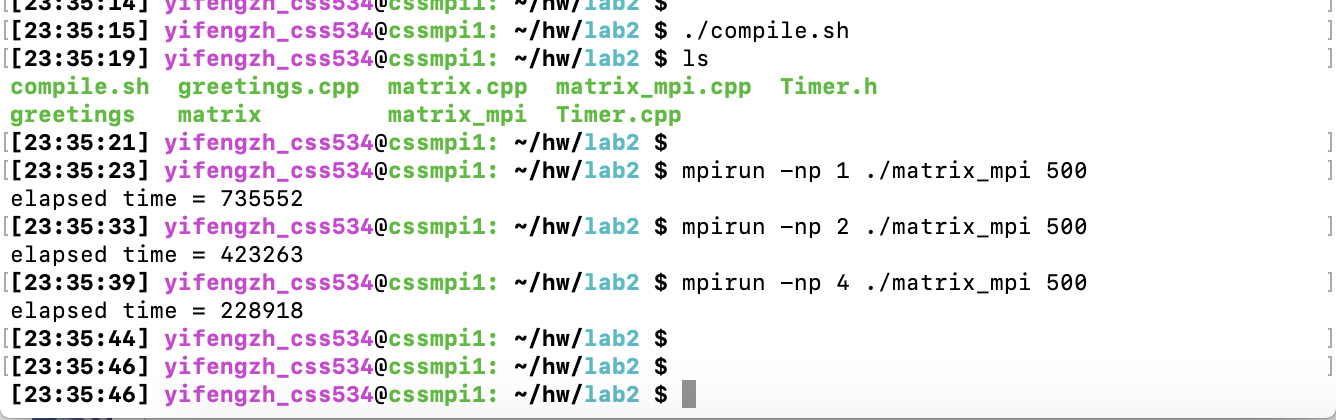
Using following method with MPI:

MPI\_Bcast(void \*buffer, int count, MPI\_Datatype datatype,int root, MPI\_Comm comm);

MPI\_Send(const void \*buf, int count, MPI\_Datatype datatype, int dest,int tag, MPI\_Comm comm);

MPI\_Recv(void \*buf, int count, MPI\_Datatype datatype,int source, int tag, MPI\_Comm comm, MPI\_Status \*status);

|  |  |  |
| --- | --- | --- |
| Node number | Elapsed Time | Performance Improvement |
| 1 | 735552 | 1 |
| 2 | 423263 | 735552/423263 = 1.74 |
| 4 | 228918 | 735552/228918 = 3.21 |



***Note****: all source code and output files in the lab2 folder.*