Homework 1

Gang Chen
2019.4.3

The code for each problem can be found in the subfolders. Problem 1 & 2 are solved by coding in C and Problem 3 is solved by coding in C++. There is a brief introduction on the top of each .c/.cpp file. The adopted method is described as follows:

Problem. 1
MPI programming:
Use MPI_SEND and MPI_RECV to implement the MPI_ALLGATHER function, and compare the performance of your implementation and the original MPI implementation.

Answer:
This is a basic usage of the MPI library. In the c code *(problem1/problem1.c), 100 times sequence of the process* is treated as a message to be sent. Firstly, all other processes send their messages to the Process 0. This forms a array along with Process 0's own message. Then Process 0 send the array to all other processes. In comparison, *problem1_allgather.c* gives the usage of the original MPI_ALLGATHER function.
The performance of the implementation is a little slower than the original function (about 10% lower).

Problem 2:
MPI programming:
Initialize a random 1024 *1024 matrix.
2.1 Implement the matrix multiplication function.
2.2 Using a 4 * 4 kernel to do the pooling operation for the matrix.
2.3 Using a 4 * 4 kernel to do the convolution operation for the matrix.

Answer:
First of all, the random 1024 *1024 matrix is represented by an array with random floats ranging from 0.f to 1.f.

2.1 **Matrix multiplication**
The implement algorithm is based on the Row-wise Block-Striped Parallel method described in the Chapter 11 of the book *Parallel Programming in C with MPI and OpenMP*. The code is *problem2/problem2-1.c*

2.2 **Pooling operation**
The stride is set to be 1 and padding is not considered. The Process 0 randomly initialize the matrix and divide the matrix into p (the number of the processes) pieces by lines. 3 more lines are added to each piece to fulfill a complete calculation. The last piece is added with 3 zero valued lines. Then the pieces are sent to all processes to do the *Max Pooling* operation. The local results are sent back later. *MPI_Scatter* and *MPI_Gather are used. The code is named as problem2/problem2-2.c.* When running the code, the number of the processors should be the divisor of 1024.

2.3 **Convolution operation**
This problem is solved similarly to 2.2. An "L" shaped 4x4 kernel is defined to do the convolution operation. The *Max Pooling* operation in 2.2 is replaced by a convolution function.

Problem 3:
MPI programming:
We will provide one folder which contains 100 small files and one folder which contains 1 big file.
Implement the wordcount algorithm respectively for the two situations and print the results to the screen.

Answer:
3.1 Word count for small files

In my code, the Process 0 scan the ".txt" files in the folder and store the paths. Then the paths are allocated by the *Block Data Decomposition* algorithm to all the processes. Every process opens several files later and count the number of the words.

The *std::vector<string> is used to store the paths conveniently.* To realize the communication of string with different size, *MPI_CHAR* type is utilized and the size of the message is the length of the string. To tell the receiver the size of a message, a message with one integer is sent before the string. The flexible size problem is solved by using this simple protocol rather than use *MPI_Probe* and *MPI_Get_count* functions (just to have a try in another way).

The number of the words and the corresponding file name is printed on the screen.

3.2 Word count for a large file

The "manager-workers" structure is adopted. Process 0 is a manager who reads the file line by line and allocates to the free workers. Other Processes play the role of worker who receive lines from manager, count for words and send counted results back to trigger another sending from the manager. When all the lines are processed, the manager sends a exit signal with the string "*sss*" to all the workers and exits itself.

In this work, the *MPI_Probe* and *MPI_Get_count* functions are applied to let the receiver know the size of a string before it is actually received.

The counting result is 14500 words when the words are split by blanks, "\n" and "\t".