Assignment 2
Tianzhu Yang
510650346

1.Problem and definition:
Assignment 2 and Assignment 1 runs the same computation, all Pairwise Computation. In assignment 1, I need to treat each row as a sequence/vector, then I can pair each other row and multiply each item in each row. Suppose a matrix has N rows of data, then there are N(N+1)/2 possible pairs, so the whole result matrix is an inverted triangle. I also need to put this inverted triangle matrix into a 1D array. However, this Assignment will be carried out in a combination of MPI and Pthread. The difference between processes and threads is that processes cannot share memory, whereas threads can. Processes can only communicate with each other by transferring data. Therefore, MPI is used. In this Assignment, I will transfer block matrices to realize the data communication between processes and calculate the results.

2.MPI and Pthreads design:
According to Assignment 1, the resulting matrix is an inverted triangle matrix. In Assignment 2, we need to transform the inverted triangle matrix into a rectangular matrix. Therefore, we need assign the data for each process. To ensure that the number of items in the matrix is N(N+1)/2, we can set the length of the matrix to N and set the width to (N+1)/2. Assuming P = N, each process will be allocated exactly one row of data. As shown in the figure below:

P_0 (0 0) (0 1) (0 2) (0 3) (0 4)

P_1 (1 1) (1 2) (1 3) (1 4) (1 5)

P_2 (2 2) (2 3) (2 4) (2 5) (2 6)

P_3 (3 3) (3 4) (3 5) (3 6) (3 7)

P_4 (4 4) (4 5) (4 6) (4 7) (4 8)

P_5 (5 5) (5 6) (5 7) (5 8) (0 5)

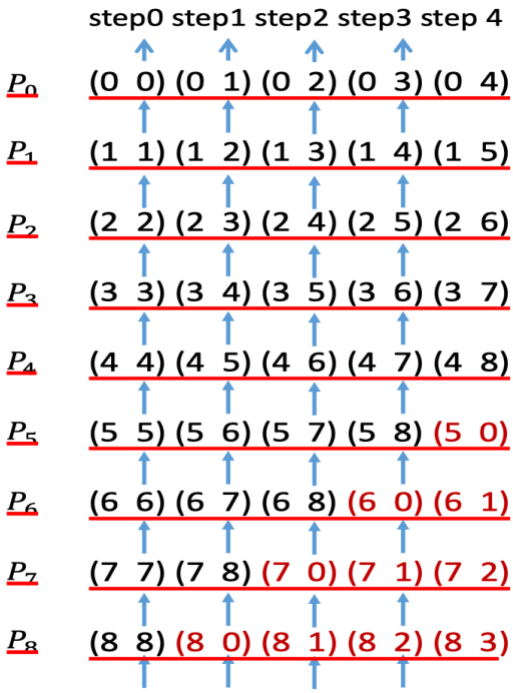
P_6 (6 6) (6 7) (6 8) (0 6) (1 6)

P_7 (7 7) (7 8) (0 7) (1 7) (2 7)

P_8 (8 8) (0 8) (1 8) (2 8) (3 8)

To create this rectangular matrix, I need to cut the items from the right of the inverted triangle to fill the items below the inverted triangle. When the second index m is greater than (N+1)/2, put the data of (0,m) in the position of (m, 0), and put the data of (1,m) in the position of (m,1), etc.

When N and process are not equal and N %P == 0, the matrix needs to be divided into P parts with N/P rows each. The resulting matrix then looks like the following:

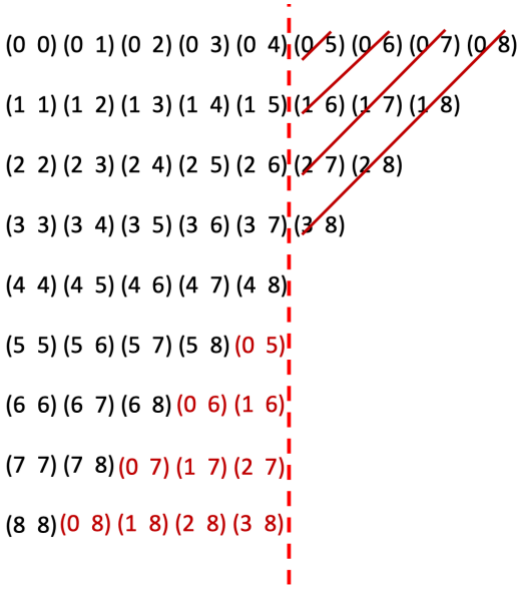


Each index here representes a block index, (index1, index2) representing the inner product of both the block of index1 and the block of index2.
Since the matrix width is (P+1)/2, I will do (P+1)/2 data transfer. At first, in each process, the block matrix 1 do inner product with block matrix 2 And then leave the block matrix 1 unchanged and transfer the block matrix 2. Then I calculate the inner product of two matrices again. After all iterations are completed, all blocks are paired, and the result is obtained.

Next, I will introduce the program to achieve the algorithm:
I divided the calculation into two parts. The first section computes the data in the first column of the result matrix. I create two sub matrices (block matrix1 and block matrix2) to store the items of the corresponding index1 and index2. I create matrix1 at process ID = 0 to store the first part of the split original matrix. Since the items of the first column are on the diagonal of the inverted triangle result matrix, so matrix1 = matrix2. After the first process completes its calculation, it sends corresponding blocks to all other processes. Then I receive the data in the corresponding process. Therefore, I used the MPI_Isend and MPI_Recv for this part. The reason for using MPI_Isend instead of MPI_Send is that MPI_Isend is non-blocking communication, which allows overlapping calculations and communication. Therefore, it avoids deadlocks.

When the process ID is greater than zero, the product of block 1 and block 2 is computed first in step 1, and the block 2 is sent to the previous process at each subsequent step. Pthread is used when block 1 and block 2 are multiplied. Multithread part is explained as follows:
result matrix of block 1 and block 2 is divided into T parts. Each thread calculates the data for the corresponding part. Load balancing is also used here. Inside each for loop, I calculate the number of remaining data divided by the number of remaining threads (assume the result is x). If x is greater than the total number of data divided by the total number of threads, then round up x, otherwise round down x. I then set up two for loops in the program. I also determine the starting position of right_block and right_block, which ensures that each thread gets the correct data.

Finally, when all processes have calculated the result matrix of the corresponding part, they send the matrix to process 0 using MPI_Isend. Process 0 Integrate all the data into the final result matrix.

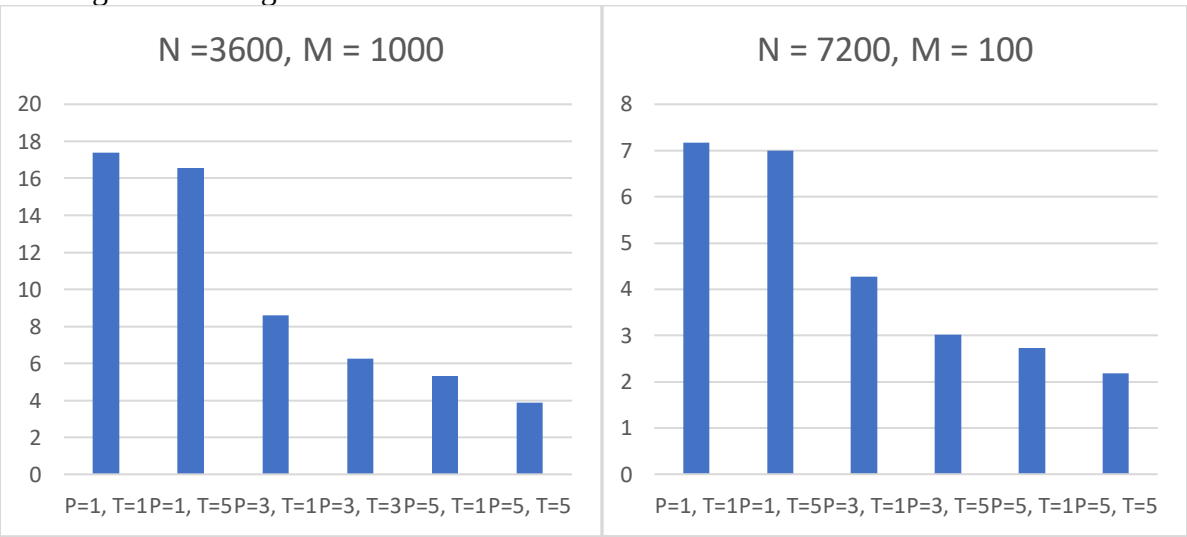


If P is equal to 9, then I get the resulting matrix to the left of the red line, I'm going to divide the matrix into the red and the black. I put the black item in the original position of the inverted triangle, and put the red item in the upper right corner of the inverted triangle. The placement method is the same as described above.

3. Testing and performance evaluation

I used the most traditional method to calculate the test result array in test function, which is same as Assignment 1, and compared the total result array with test result array. This can be used to check for data errors.

Then I use the method of control variables to check the effect of P, N, M, T. To see if MPI communication is more efficient than single-process + single-thread, I leave the amount of data unchanged and change P and T and check the Runtime.



The followings are conclusions from the image: 1. When other variables (N,M,T) remain unchanged, increasing the number of processes can improve efficiency. 2. When other variables remain unchanged, increasing the number of T can also improve efficiency.

4.Discussion:
From the previous images, we can conclude that using MPI for multi-process communication is more efficient than using a single process. The reason is that different processes exchange data through MPI, allowing different processes to handle different tasks. You can also use multiple threads in each process to improve efficiency.

5.Known issues in programs:
Previously, I used MPI_SEND and MPI_RECV for data communication between processes. The program could not run if there was too much data. The reason is that MPI_Send uses blocking communication. MPI_Send is blocked until the cache is empty. When the amount of data is too large, it is easy to have deadlock. At this point, I changed MPI_Send to MPI_Isend, which is non-blocking communication. It can avoid deadlocks.

Manual:
Build the MPI.c file: make
Remove the running file: clean
Run MPI.c file: mpirun -np P MPI N M T (you can assign N, M, P, T with different numbers)