COMP 422_ Assignment 3 Zhipeng Wang

Program Descriptions:

This program employs the MPI to implement a parallel solving of the matrix product problem. The program uses the 2.5 D matrix multiplication algorithm (see the reference written by Edgar Solomonik, James Demmel, Technical Report No. UCB/EECS-2011-10).

Basically, this algorithm combines the 3D matrix product algorithm and the Cannon matrix product algorithm. By distributing the matrix along the Z direction (where c sits in), it can reduce the entire complexity cost of the matrix product.

The program I wrote have borrowed some codes of 3D and Cannon programs from the textbook (Ananth Grama, Anshul Gupta, George Karypis Vipin Kumar <Introduction to Parallel Computing>) and uses them to built the new 2.5D version of matrix product.

The program includes the following steps:
1. Initializing of the matrix A, B and C;
2. Broadcasting of the A and B along the Z direction;
3. Doing the shift of matrix A and B at each 2D plane of a specific Z coordinate;
4. Doing the equivalent steps of Cannon Algorithm at each 2D plane to shift and multiply the matrix;
5. Reduce the result along the Z direction back to the front face, where the matrix C sits in.
6. Compare and print out the result.

The dimension of the matrix and the value of c should be provided as the two arguments when running the program.

Below is the performance table with a variety of test cases:

P	C	Performance (s)
1*4*4	1	296.751524
4*4*4	4	77.518367
2*4*4	2	148.375373
2*6*6	2	62.006664
2*8*8	2	43.600062
3*6*6	3	44.356002
3*9*9	3	18.048096
4*8*8	4	22.002799

Discussion about The bandwidth and latency lower bounds:

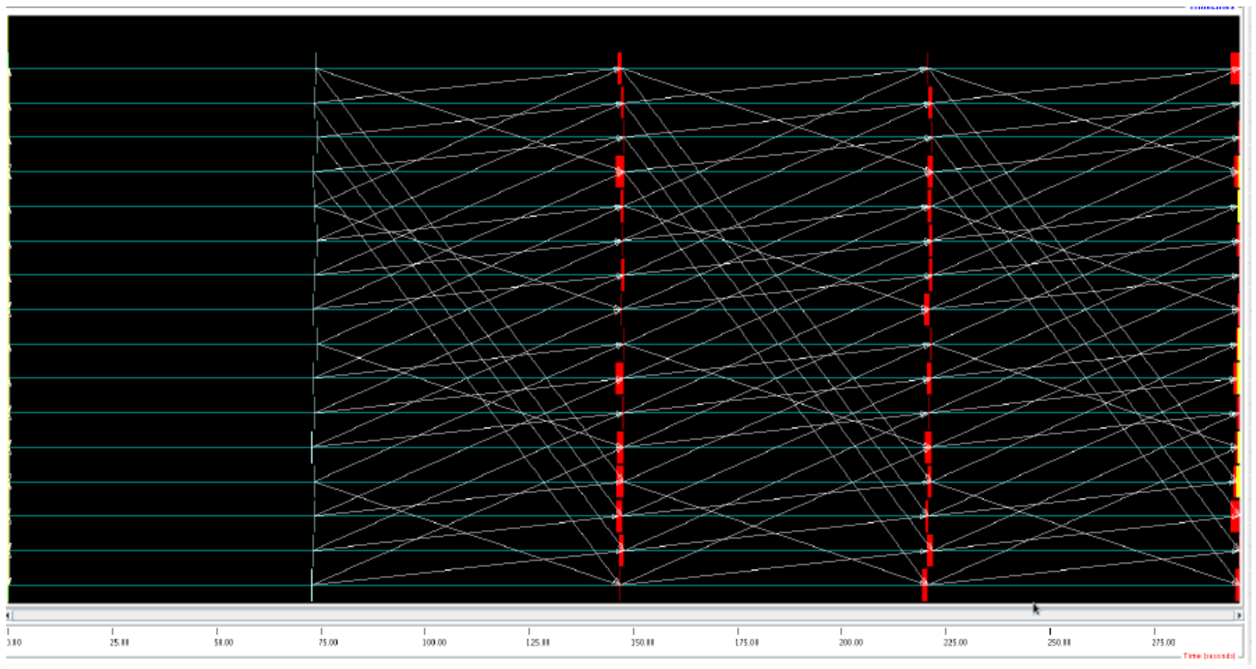
As described in the referenced literature, this algorithm "achieves the 2.5D bandwidth lower bound and gets within a factor of O(log p) of the 2.5D latency lower bound". By making use of the Cannon Algorithm for each portion of the matrix, the algorithm initially shifts the matrix A and B with the information from the Z direction. This shift is different for different 2D plane along a specific Z coordinate. The bandwidth cost is W = Ω(n^2/sqrt(cp)) and the latency cost is

Ω(p^0.5/c^1.5) as shown in the reference. In the extreme case of c = 1, we reproduce the Cannon algorithm, where the bandwidth cost and the latency cost is high, i.e, we increase the communication cost. However, in this case, we reduce the memory cost. In the other extreme, we reproduce the 3D version of matrix product. The communication cost is reduced while the memory cost is high. So the 2.5D version of matrix product takes advantage of both the communication and memory cost in order to improve its performance. By adding the copies along the Z direction, we use more memory compared with Cannon Algorithm, while reduce the communication cost since the looping reduce from sqrt(p) to sqrt(p/c^3),where p is the number of available processors.

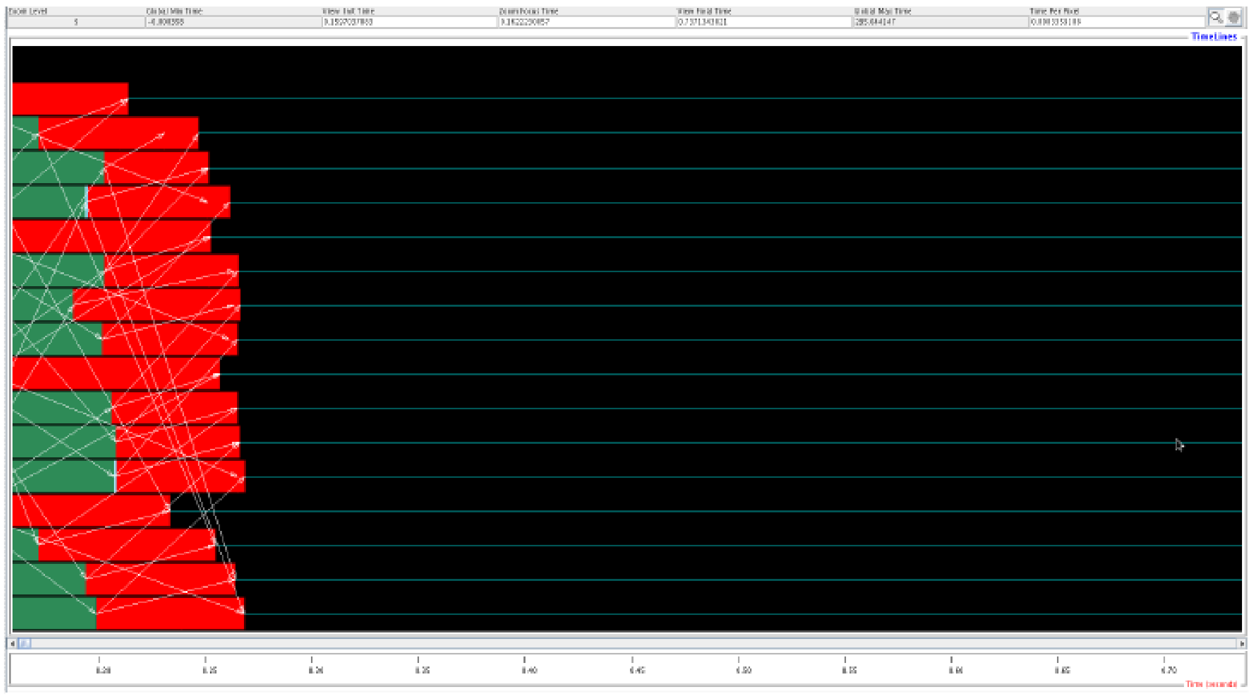
Discussion of Collective Communication:

In the first step of our program, the processors in the front face (Z=0) broadcast the matrix A and B along the Z direction, which is the One-to-All Broadcast. Then the shifts for the equivalent Cannon Algorithm are point-to-point communication. The final step of reduction towards the Z=0 plane is the All-to-One Reduction.

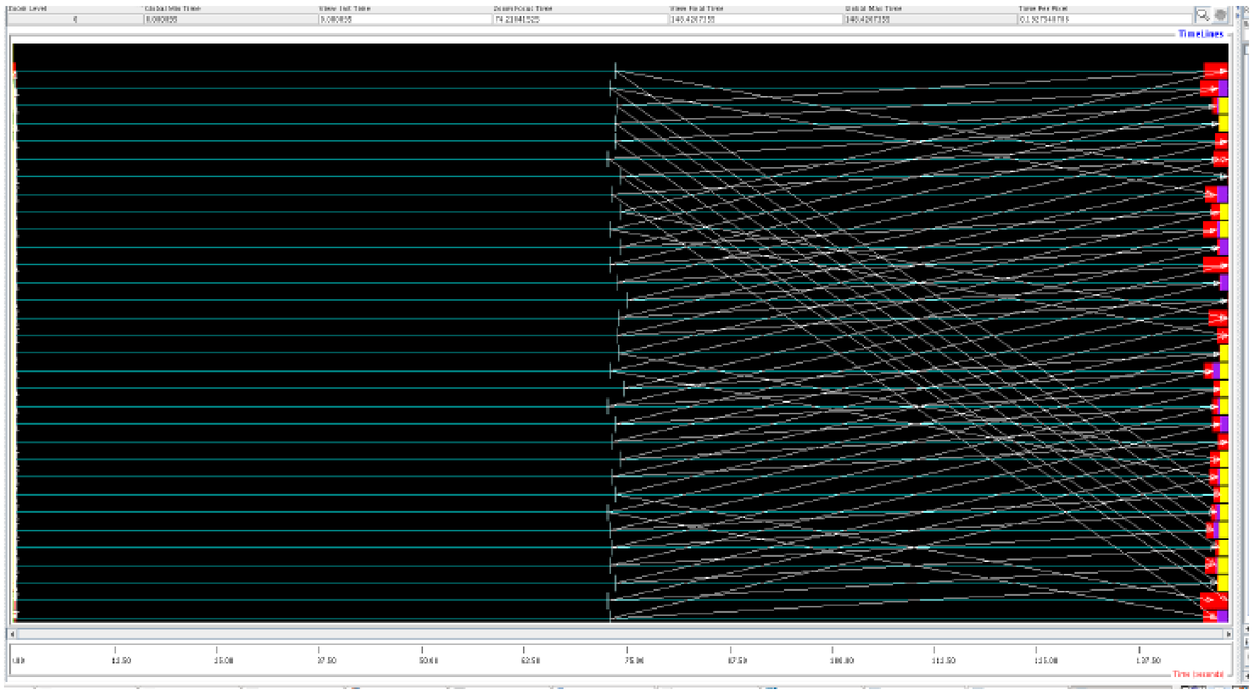
Below is the Screenshot of Jumpshot:



The screenshot is for p=1*4*4 case. We can see here there are four steps in this calculation. The first one is the first matrix multiplication before the Cannon Shift. The other three are for the Cannon shift and matrix calculation. Also, we can tell from here the Non-blocking sending and receiving messages. The communication wait, as shown in the screenshot, is very shot compared with the time in the effective calculation. The shift at the beginning is very short. We expand it to be like below: we can see it takes no more than 0.3 seconds. The barriers, as shown in the yellow trunks in the end, are also very short.

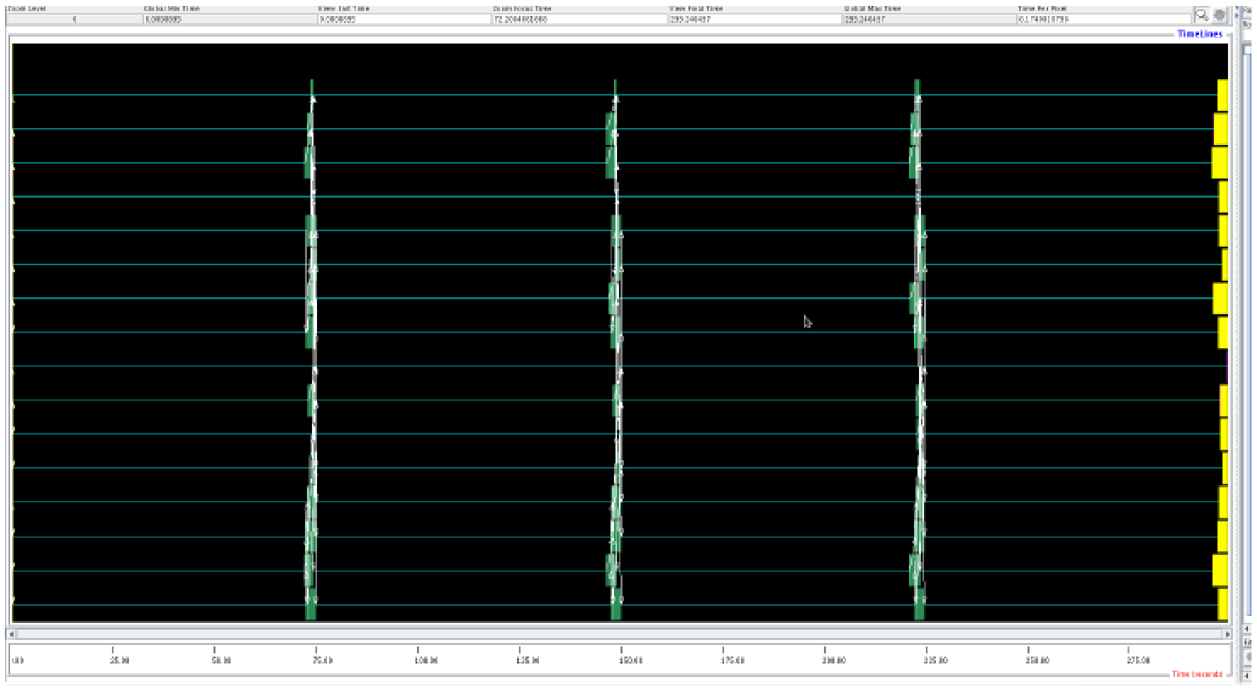


Screenshot for the case of 32 processors (2*4*4) and c=2 is shown below:



The purple trunks represent the reduction, when c>1, there would be the final reduction communication involved.

Jumpshot for the case of blocking communication (1*4*4 and c=1)



From the above screenshot we can see the difference between non-blocking and blocking communication. The non-blocking communication can perform communication and computation almost simultaneously, which would be a huge advantage for speeding up the process.