Brief Report for Assignment 4

1. Illustration of correctness of code:

In assignment 4, three methods to multiply two matrix are implemented, those three methods are sequential approach, static parallel approach and dynamic mapping approach. In order to clarify the correctness of code, a simulation with two matrix (both 4 rows and 4 columns) multiplying are executed, result is shown below:

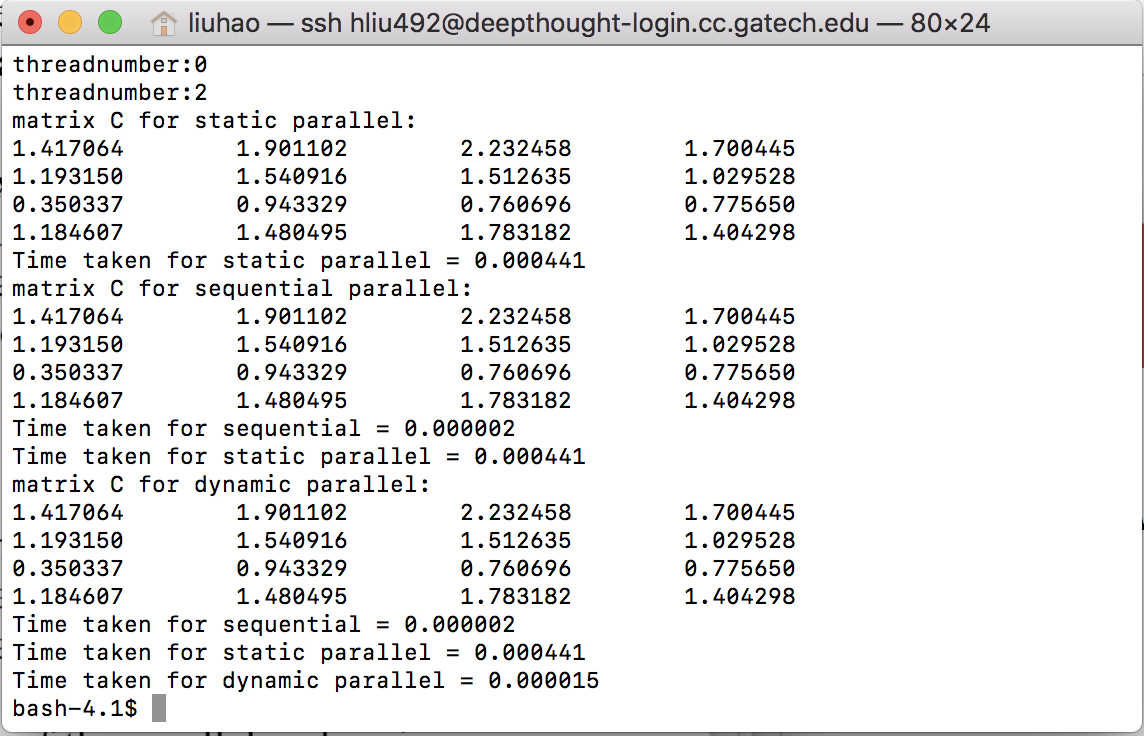


Figure 1. results for 4\*4 matrices

As it shows, all three methods got the same right answer.

1. Simulation results and analysis:

In order to get the relationship between run time of these three methods and the size of matrices, a series of simulation were implemented. Results is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| size | sequential | static | dynamic |
| 4 | 0.000002 | 0.000441 | 0.000015 |
| 50 | 0.001482 | 0.001153 | 0.000482 |
| 300 | 0.317256 | 0.091197 | 0.051634 |
| 600 | 2.140409 | 0.557245 | 0.346976 |
| 900 | 11.249635 | 4.485673 | 0.412011 |

Table 1. runtime of three methods when size is changing

As it shows , when the size of matrix is very small, sequential method performs best, however, when the size of matrices grow to about 50, runtime of sequential method and static parallel approach are close and runtime of dynamic approach remains low. When the size of matrices grow to a relative large scale, sequential methods tend to be the worst method, it’s runtime is much larger than the other two approaches, mean while, dynamic approach is always better than static approach in these cases. This might because, when size is small, the weight of runtime for processors visiting shared variables are larger than the runtime exactly used for calculation, thus sequential methods perform better in this case. However, when size grows larger, runtime are saved for parallel approaches since tasks are divided into several parts. As for the comparation between static approach and dynamic mapping approach, since for the later method, processors with faster speed will process more tasks and thus the total runtime for dynamic method is lower than static one.

In order to get the relationship between number of thread and speedup of these two parallel approaches, several experiments were implemented. N is the number of threads, speedup is the ratio of parallel method’s runtime to sequential method’s runtime. Results are shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Sequential  runtime | Static runtime | Dynamic runtime | static speed up | dynamic speed up |
| 4 | 11.243326 | 3.12003 | 1.92613 | 3.603595478 | 5.837262282 |
| 6 | 11.42995 | 2.309925 | 1.373519 | 4.94818966 | 8.321651903 |
| 8 | 11.494319 | 2.058991 | 1.142418 | 5.582500846 | 10.06139522 |
| 10 | 11.41095 | 1.745957 | 0.925744 | 6.535640339 | 12.32624462 |
| 12 | 8.261554 | 1.597292 | 0.549879 | 5.172225241 | 15.02431262 |

Table2. results on matrices size of 900\*900

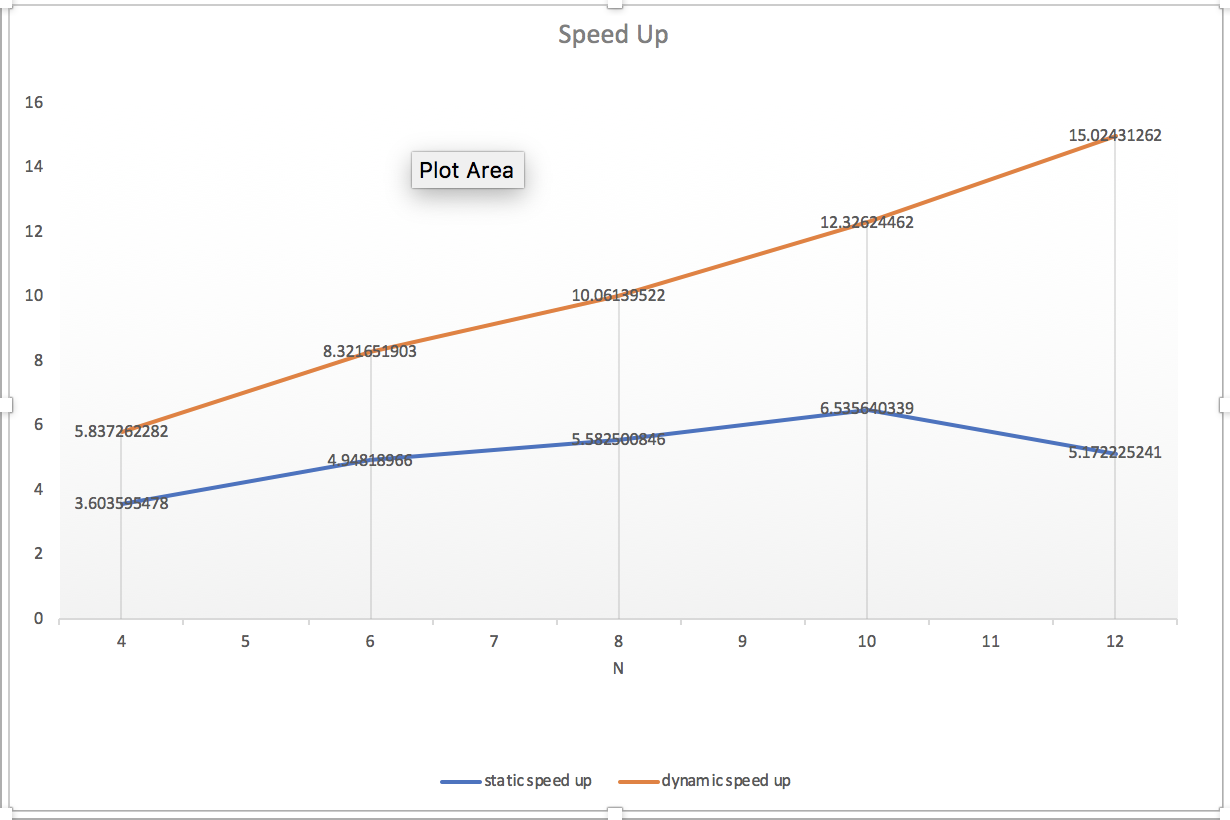


Figure2. results on matrices size of 900\*900

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Sequential  runtime | Static  runtime | Dynamic  runtime | static speed up | dynamic speed up |
| 4 | 0.00253 | 0.001374 | 0.000645 | 1.841339156 | 3.92248062 |
| 6 | 0.002546 | 0.001491 | 0.000391 | 1.707578806 | 6.511508951 |
| 8 | 0.002525 | 0.001658 | 0.000383 | 1.52291918 | 6.592689 |
| 10 | 0.002538 | 0.002011 | 0.000387 | 1.262058677 | 6.558139535 |
| 12 | 0.002535 | 0.002224 | 0.000458 | 1.139838129 | 5.534934498 |

Table3. results on matrices size of 50\*50

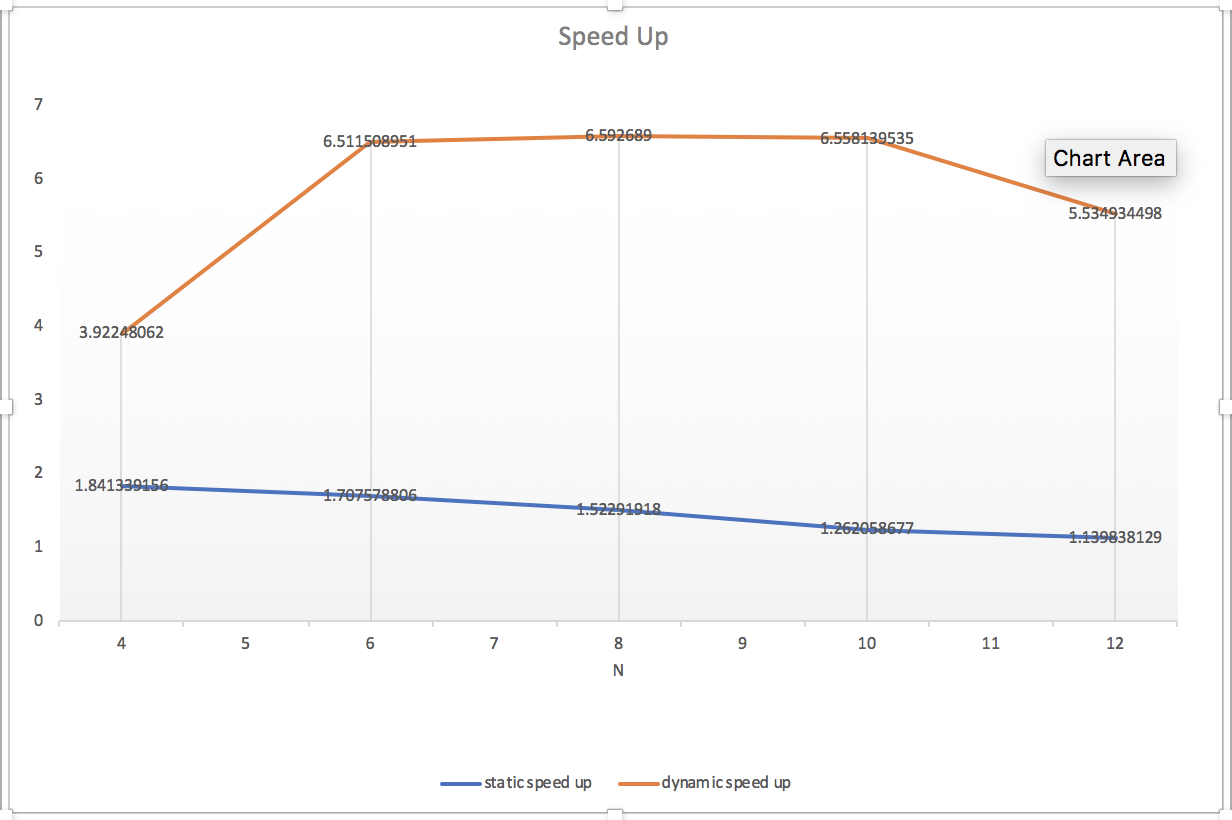


Figure3. results on matrices size of 50\*50

As above results show, when the size of matrices are 900, the speedup of both two parallel methods increase with the increase of thread number, when the size of matrices are 50, the speedup of static method decrease with the increase of thread number mean while the speedup of dynamic approach first increase then decrease. The result shows that parallel methods will have better performance when the scale of problem is big.

1. Literature search:

MPI is the short of “Message Passing Interface”, which is a tool customers can use to do parallel computing. In MPI, each processor has rank and processors can communicate with each other using function defined in MPI.

For the algorithm calculating multiply of two matrices using MPI, several functions should be used, including MPI\_Init(); MPI\_Comm\_rank(); MPI\_comm\_size(); MPI\_Send(); and MPI\_Recv();. The first function is to Initialize the environment for MPI, the second and third function each get the rank of current processor and the total number of ranks. The last two function involve the communication between porcessors.

Below is the algorithm used to compute matrix(C = A \* B ) multiplication using MPI:

1. every processor initialize with necessary local memory to store variables
2. In main processor( for example rank 0), matrix A are divided by rows and each is sent to other porcessors by using MPI\_Send(). Also the Matrix B are sent to other processors too.
3. Other processors use MPI\_Recv() to get the task rows of A and Matrix B, and then calculate the corresponding rows of C. After that, each processor use MPI\_Send() to send the result rows of C back to the main processor.
4. Main processor use MPI\_Recv() to receive the results sent by other processor and combine them into the shared variable C[][]. Return C.

Reference:

1. <https://computing.llnl.gov/tutorials/mpi/>
2. https://blog.csdn.net/lcx543576178/article/details/45892839