Homework2: Permutations and Sort

ECE 5720

Introduction to Parallel Computing

Name: Yanling Wu

NetID: yw996

Way1: Sort by the maximal value of one column

The effect of the way to code:

Decide how to find the maximal value of one column in submatrix and how to swap the rows are vital to solve this problem. There are three ways that we could use.

1. Totally sequential to find the maximal and swap the rows;
2. Totally parallel to find the maximal and swap the rows;
3. Mixed to use the sequential and parallel way to find the maximal and swap the rows.

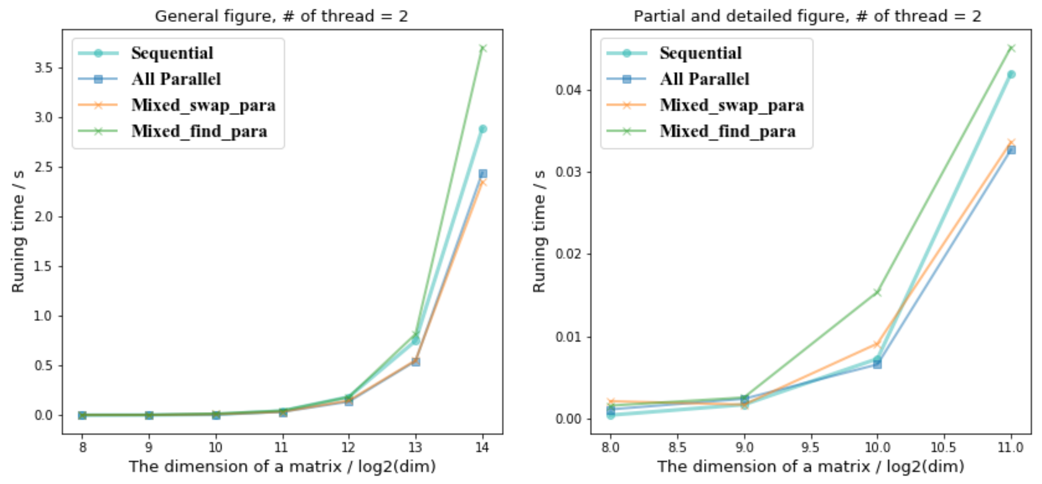
I wrote codes using these three ways, explored and analyzed which way is the best way to do this job.

Firstly, the left figure of Fig 1 below shows that the how the running time changes as the dimension of matrix increasing when the number of threads is fixed to 2; the right figure of Fig.1 below shows the part of left one and amplified the front part of the left one. The reason to use 2 threads is that after analyzing the relationship between running time and the number of threads, I found that when the number of threads is 2, the running time is the fastest.

From this figure, we can know several things:

1. When the dimension of the matrix is small, such as less than 2^{10} , the sequential way is the fastest way to sort and swap the matrix. But when the dimension of the matrix continuously increasing, the performance of sequential way got worse and the performance of parallel way and the mixed way where swap function is used parallel way and the find maximum is used sequential way got better.
2. Comparing the two ways of mixed with sequential way, when "find maximum value function" is sequential and the swap is parallel, the running time is the smallest and interestingly, when "find maximum function" is parallel and the swap is sequential, the running time is the longest. This is because the fast way to swap is to use parallel and fast way to "find maximum value function" is sequential. The reason of this might be the way I wrote code.

Fig 1. Way1 runntime by different ways



In order to parallelly to find the maximum value of one column, I set the shared variable maxVal and maxRow to mark the current maximal value and according row number, which means when some threads want to change the maxVal and maxRow, other threads must wait for the one that is writing them and thus, the running time will become longer. And when the dimension of the matrix is small, this kind of wait will slow down the running time. That is why when the dimension of matrix is small, the sequential way is faster.

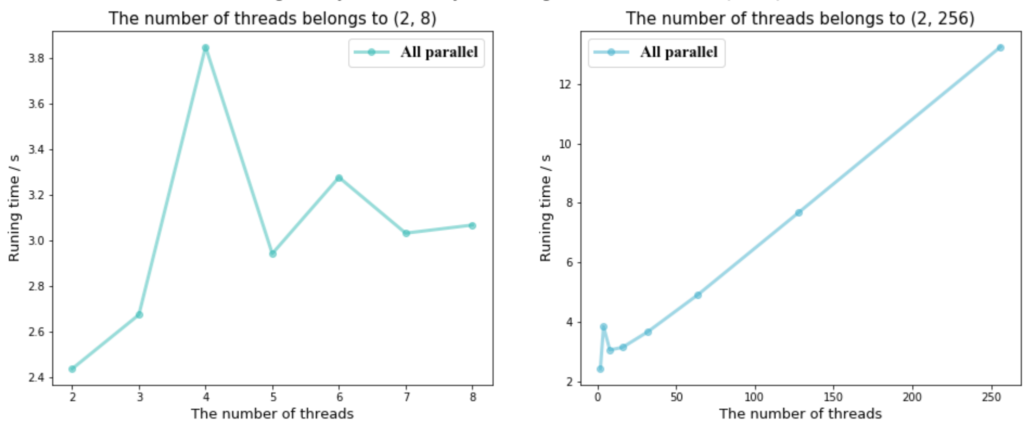
```
void final_A(int dim) {
    int i = 0, j = 0;
    int maxVal = -1;
    int maxRow = 0;
    for (j = 0; j < dim; j++) {
        maxRow = 0, maxVal = -1;
        #pragma parallel omp for schedule(static) shared(maxRow, maxVal)
            for(i = j; i < m; i++) {
                if (maxVal < *(A + n * i + j)) {
                    maxVal = *(A + n * i + j);
                    maxRow = i;
                }
            }
        swap(j, maxRow);
    }
}
```

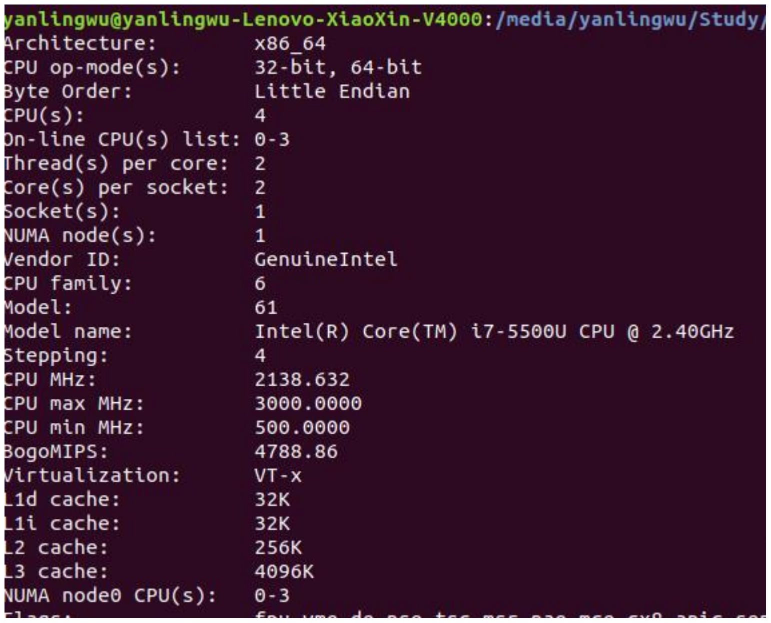
The effect of the number of threads:

After knowing the best way for matrix is to sort it, I also explored what effect the number of threads to the running time. The figure below shows the relationship between running time to the number of threads. The left one only opens 2 to 8 threads which is my computer is 4 cores and each core can have 2 threads. My computer's detail information shows below. And the figure shows that when opening 2 treads, the running time is the shortest and when opening 4 threads, the running time is the longest. Besides, when increasing more opening threads, the running time will increase sharply.

All in all, if the dimension of the matrix is less than 2^10, the best way to sort is sequential to run. And if the dimension of the matrix is more than 2^10, the best way to sort is parallel to run.

Fig 2. Way1 runntime by increasing # of threads, dim = (2^14)




```
yanlingwu@yanlingwu-Lenovo-XiaoXin-V4000:/media/yanlingwu/Study/
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 61
Model name:            Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz
Stepping:              4
CPU MHz:               2138.632
CPU max MHz:           3000.0000
CPU min MHz:           500.0000
BogoMIPS:              4788.86
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              4096K
NUMA node0 CPU(s):     0-3
```

Speed_up:

# of threads	2	3	4	5	6	7	8
Totally parallel	0.3639	0.705	1.105	1.284	1.347	1.374	1.183
Only Swap parallel	1.191	1	0.7972	1.259	1.256	1.356	1.231
Only find max parallel	0.257	0.659	0.472	0.929	1	0.917	0.779

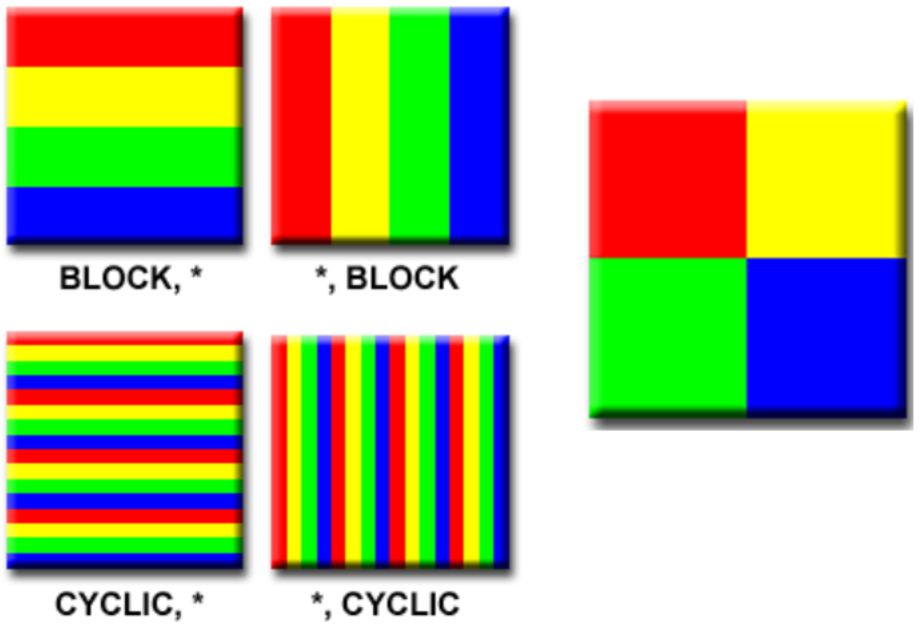
Way2: Sort by the maximal value in submatrix

Comparison five ways:

For finding a maximal in absolute value elements in a 2D, there are several ways to decomposition data:

1. block row
2. block column
3. cyclic by row
4. cyclic by column
5. tile

According to my understanding, the specific ways to assign to threads shows below. The upper two figures show the block way, and the lower two figures show the cyclic way. The tile way is like the right figure, the different color represents one tile block and there will be many threads to sun in each tile block.

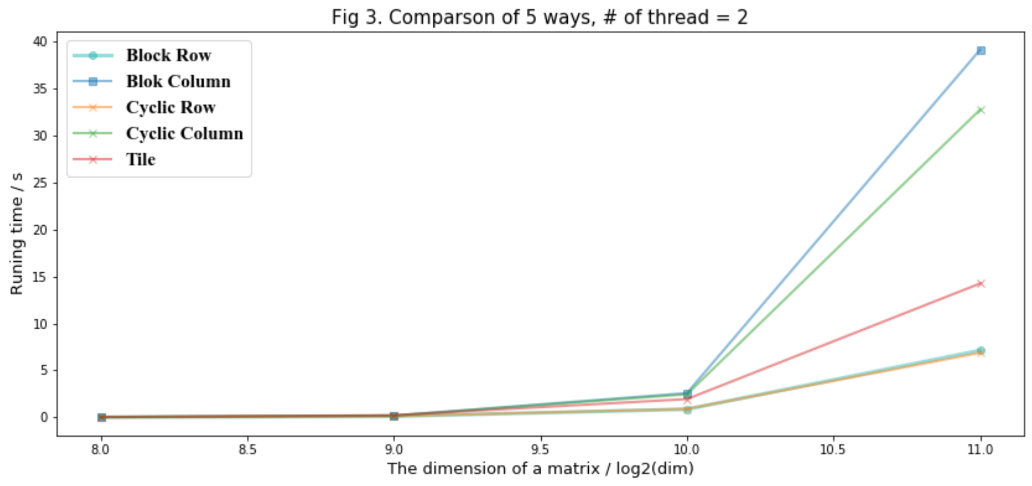


In order to find which way is the best way to decompose data, I actually wrote these five ways to compare and analyze them. According to the way1, we know that the best performance for my computer will be gotten when the number of threads is 2. So, I used 2 threads to run these five codes and got the figure below.

1. From this figure, we could know that the block column way is the slowest way to solve the second problem and the cyclic column is the secondary worst one. This might be because I used 1D pointer array to store the matrix in order to improve the performance and for array, the address of every row is neighborhood, and the address of every column is not. So, if threads are

assigned by column, then the different threads will jump to read data instead of moving to close one which will spend more time.

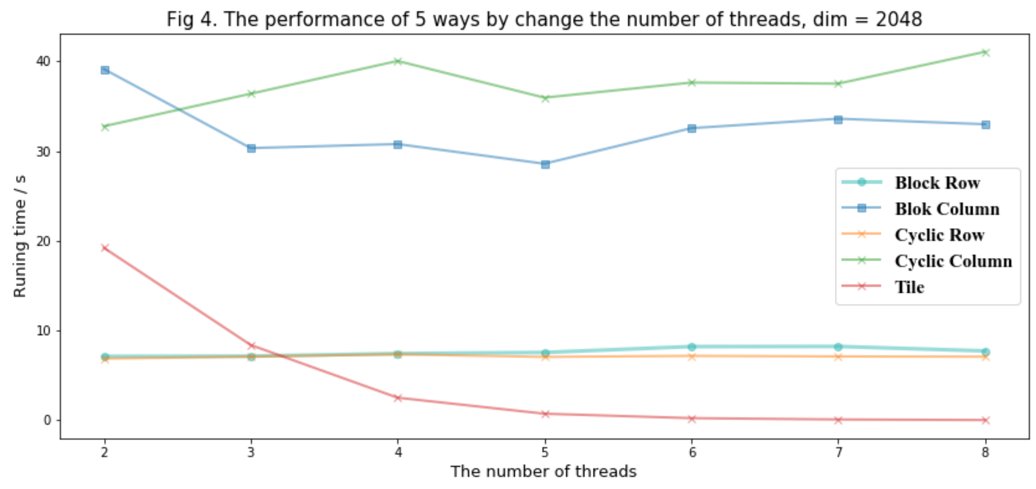
2. The running time of block row and cyclic row is very close. But according to the actual data, the performance of the cyclic row is better. Because the speeds of different threads are different, some threads run faster, some run slowly. And the cyclic row way can dynamically assign the task to the threads that have already finished the current job. First finish first start new job. But the block row way will assign every thread same workload, which means if some threads finished earlier, they will do nothing but wait for slower threads to finish the task and it will waste some time, so block row will be slower a little bit than cyclic row way.
3. About the tile way, it is somewhat tricky to test. For figure below, I chose the tile size equals to half of the dimension of the matrix and also fixed the number of threads. So, the performance of tile way is not bad and not good. In fact, the performance of tile way depends on the value of tile size and the number of threads. I will discuss this next.



The number of threads' effect to performance:

I opened 2 to 8 threads in my computer and compared the performance of different number of threads and ways to implement code and results show following.

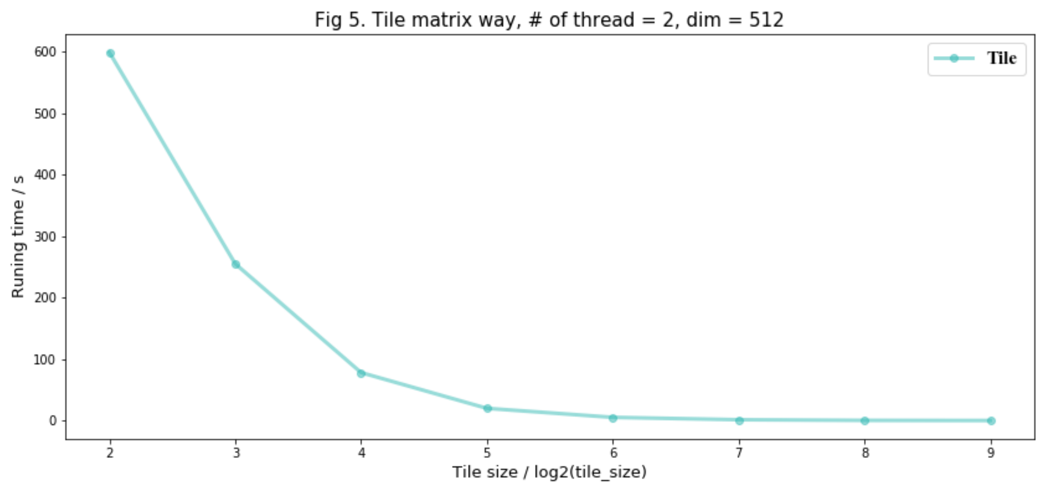
1. No matter the number of threads, the performances of the block column and the cyclic column are not good. They spend more time.
2. The performances of the block row and cyclic row are very stable, which means the performance almost have little relation to the number of threads.
3. For tile way, I also use half of dimension of matrix as my tile size, and the matrix will be tiled to four parts. And the performance of tile way is getting better as the number of threads increasing.



The discussion of Tile matrix way:

After the exploration above, we have known that the cyclic row and block row are very good. But the tile way is very tricky. And the performance of the tile way counts on the number of threads. Besides, there are one significant factor that will have huge influence on the performance of tile way --- the tile size.

Next, I explored how the tile size affects the performance of this way. I fixed the number of threads to 2, and the dimension of matrix to 512 * 512.The figure below shows that the performance of the tile way will get better as the size of tiling increasing. I think the reason of this might be that if the tiling size is bigger, it will be close to block way so the running time will decrease. But I am confused about why the performance of tile will overcome the block and cyclic row way when the threads increasing.



Overall, for the problem 2, the performance of cyclic row way is the best and the most stable. Although in some situation, the performance of the tile way will be better than that of cyclic row way, it is difficult to find a good parameter to make the performance of tile way best. Hence, I will choose the cyclic row way to solve this problem.