TIC4005

Project 2

Zheng Shijie A0177971M

Content

1. General Description

A) Overview

The whole program is able to solve N queens attack problem in an efficient way. Basically, the idea is to list down all the possibilities and check whether each possibility satisfies the requirement. But the traditional way such as using int type to represent the chessboard is too cost on space. And also it will slow the calculation speed when the number of possibilities gets larger. Thus, the solution in this document described is optimized from two perspectives compared to the traditional solution. The program is not only optimized from the perspective of the algorithm, but it also optimized from the perspective of data structure.

B) Data structure & algorithm description

In the program, each piece of the queen is represented by a bit, and the chessboard is represented by few bytes which have at least n*n bits. Since in the memory, the bit is the smallest unit to represent a value and C has the instructions to operate bits, using bit operation to solve this problem is the most approach to saving space and calculation compared to other data structures.

Basically, the program algorithm works in the following way:

1. As 1 means existing queen and 0 means no queen, all possibility combination is the value of bit representation from all 0 to all 1. For example, on a 4*4 chessboard, all the possibilities are 0b0000 0000 0000 0000 to 0b1111 1111 1111 1111. To go through all the possibilities, the way is to add 1 from the beginning until all bits become 1. Through this approach, it helps to list down all possibilities in a simple way.

2. For each possibility, the program does a check on all the pieces of queens the chessboard has and verify whether it satisfies the condition. If not, the result will be discard, otherwise it will be saved.

3. Repeating above algorithm until all possibilities are verified.

4. Find the maximum pieces of chess among all the possibilities.

5. Output the result

Based on this algorithm, the parallel part is mainly done as the following:

The first parallel part in the program is on checking tasks.Since every chessboard are need to be checked

and these tasks can be assigned to different process to do, I created a variable called threadID to help on distribute work equally. Basically the idea is that each process will only check the chessboard when the threadID is matching with process rank. The threadID is a variable that greater or equal than 0 but less than the number of process. It will be added 1 when every time adding a piece of chess to the chessboard. If the threadID is greater than the number of process, it will be reset to 0. So by conducting this, each process will be balancedly assigned the task of checking chessboard.

```
298
299     while (pieces < size)
300     {
301         AddPieceToBoard(board, blocks);
302         pieces = GetPiecesCount(board, blocks, size);
303
304         if (threadId == procid)
305         {
306             BOOL result = CheckBoard(board, size, side, attack, wraparound);
307             if (result == TRUE)
308             {
309                 boardsCount++;
310                 boards = (BOARD *)realloc(boards, boardsCount * sizeof(BOARD));
311                 boards[boardsCount - 1].pieces = pieces;
312                 boards[boardsCount - 1].board = InitializeBoard(blocks, size);
313                 memcpy(boards[boardsCount - 1].board, board, blocks * sizeof(BYTE));
```

Figure 1.B.1

```
323                 printf("pieces:%lld\r\n", pieces);
324                 printf("----------------------------------------\r\n");
325 #endif
326         }
327
328         if (++threadId >= numprocs)
329         {
330             threadId = 0;
331         }
332     }
333
334     MPI_Barrier(MPI_COMM_WORLD);
335
```

Figure 1.B.2

The second parallel part in the program is on finding the maximum pieces among all available possibilities. Each process will find the maximum pieces in their own results and send this result to process 0,which is the master process. The process 0 will receive all the results sent from other processes and find the global maximum pieces among all process results. Then the process 0 will broadcast this result to all other processes. Other processes will use this global maximum pieces to check whether it should output the result.

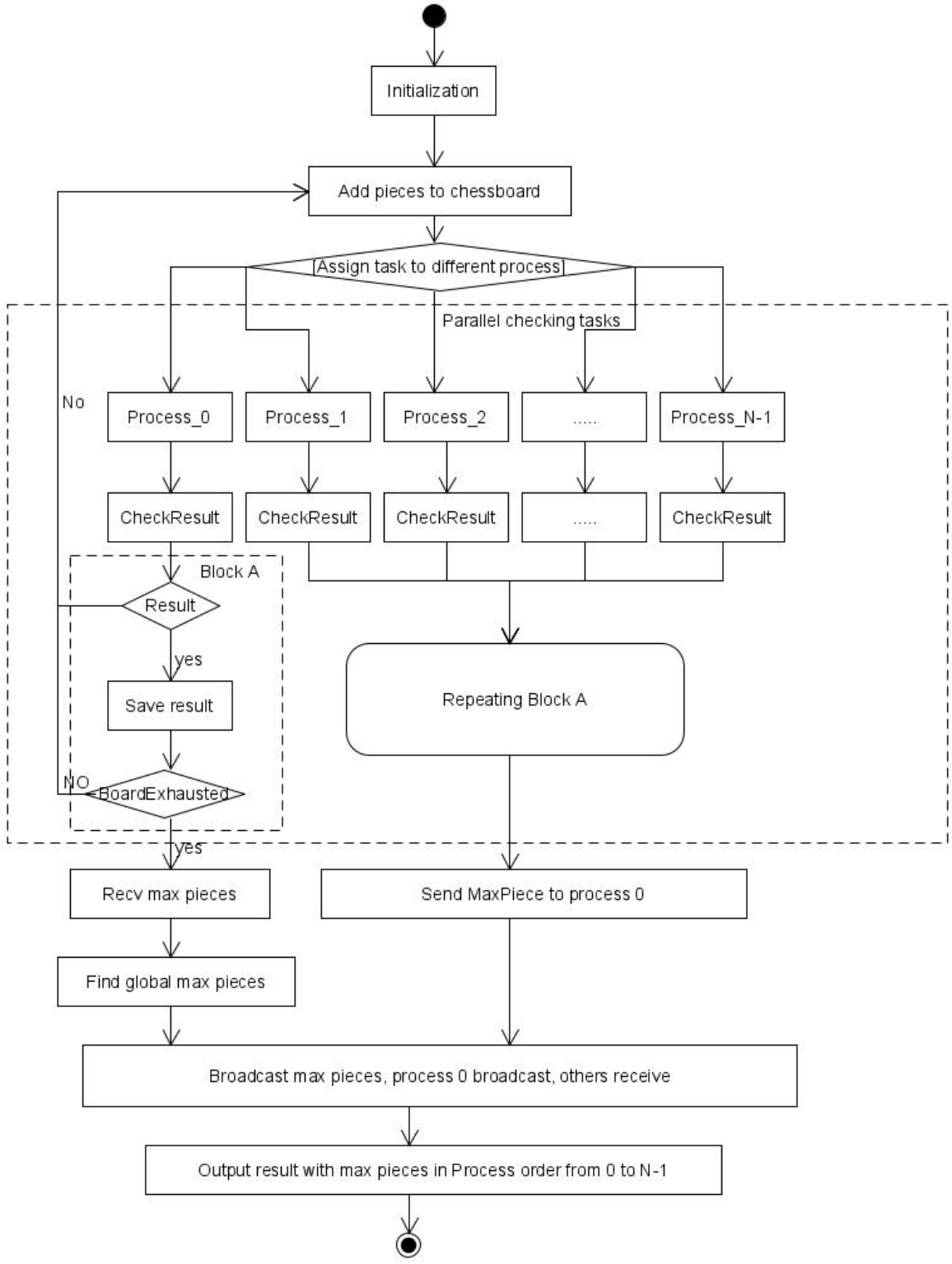
```
if (procid == MPI_MASTER_PROCESS)
{
    //Waiting for result
    // int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm, MPI_Status * status)
    int i = 0;
    for (i = 1; i < numprocs; i++)
    {
        ULL max = 0;
        MPI_Status status;
        int r = MPI_Recv(&max, 1, MPI_UNSIGNED_LONG_LONG, i, MPI_TAG_MAX_PIECES, MPI_COMM_WORLD, &status);
        if (r != MPI_SUCCESS)
        {
            fprintf(stderr, "MPI_Recv with error: %d.\r\n", r);
        }

        if (max > maxPieces)
        {
            maxPieces = max;
        }
    }
    _p("Final max pieces:%llu\r\n", maxPieces);
}
```

Figure 1.B.3

```
361     }
362     else
363     {
364         //Sending result to process 0
365         // MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
366
367         int r = MPI_Send(&maxPieces, 1, MPI_UNSIGNED_LONG_LONG, 0, MPI_TAG_MAX_PIECES, MPI_COMM_WORLD);
368         if (r != MPI_SUCCESS)
369         {
370             fprintf(stderr, "MPI_Send with error: %d.\r\n", r);
371         }
372     }
373
```

Figure 1.B.4

The working flow chart is shown as below:



C) Optimization

1. Data representation optimization(space and calculation optimization):

At the beginning to solve this problem, the int type was used to represent each piece of chess. And the int array is used to represent the chessboard. This tech takes a lot of memory and calculation resources. The current representation, which is using bytes and bit, saves a lot memory and also expedite significantly on calculation speed.

2. Output sequence optimization:

Due to each process will output the result by themselves, this caused a messy problem that the result is not print with correct format. In order to solve this, MPI_Send and MPI_Recv are used to force all processes output result sequentially. The idea is to utilize the blocking communication feature of MPI_Send and MPI_Recv. Each process will not output the result until receiving the last process's message, except the process 0 .

```
382     if (procid == MPI_MASTER_PROCESS)
383     {
384         ExportResult(boards, maxPieces, boardsCount, procid, numprocs);
385         MPI_Send(&exportResult, 1, MPI_INT, procid + 1, MPI_TAG_EXPORT_RESULT, MPI_COMM_WORLD);
386     }
387     else
388     {
389         if (procid != numprocs - 1)
390         {
391             MPI_Recv(&exportResult, 1, MPI_INT, procid - 1, MPI_TAG_EXPORT_RESULT, MPI_COMM_WORLD, &status);
392             ExportResult(boards, maxPieces, boardsCount, procid, numprocs);
393             MPI_Send(&exportResult, 1, MPI_INT, procid + 1, MPI_TAG_EXPORT_RESULT, MPI_COMM_WORLD);
394         }
395         else
396         {
397             MPI_Recv(&exportResult, 1, MPI_INT, procid - 1, MPI_TAG_EXPORT_RESULT, MPI_COMM_WORLD, &status);
398             ExportResult(boards, maxPieces, boardsCount, procid, numprocs);
399         }
400     }
```

Figure 1.C.1

2. Evaluation

A) Evaluation process description

The parallel efficiency is evaluated based on comparing the execution time. The evaluation process is that based on different numbers of cores and the value of k, comparing the execution time on the different value of N.

Due to the time limitation, currently the data obtained is only for N from 3 to 5, k from 3-8 with different number of process, which is 2,4,8 ... til 64. For the rest such as N is 7,8,9 etc is still on running, thus they are excluded from this report for discussion.

The table and graph is shown in the section C and section D. Basically, we can see that the efficiency of parallel is not positively related with the number of processes. There is always a most efficient balanced point between the number of process and the size of the problem.

For N= 3 and N=4,the graphs are similar. Both of them have a higher time cost when the number of process is 64. According to the algorithm stated before, the more processes initialized, most of the processes are more in idle status(It means just adding the pieces to the chessboard but not doing checking work). When the problem size is small, the effects of communication consumption between processes is amplified. This is why when the number of processes is 64, the time is used more than when it is 16 or 8.

There are two peak in the graph of N =3 and N=4 at k=3, this is because when k=3, there are more chessboard satisfied the condition and it takes more time to checking the whole chessboard.

For N=5, the graph is like a reverse bell-curve. The reason why time consumption from the 8 number of processes to the 2 number of processes goes higher is the 8 processes has a higher calculation ability to speed up the parallel calculation than what the 2 processes has.

B) Communication Pattern

The communication pattern is analysis based on 8 processes with parameter N=4, k =3, l=1,w=0.

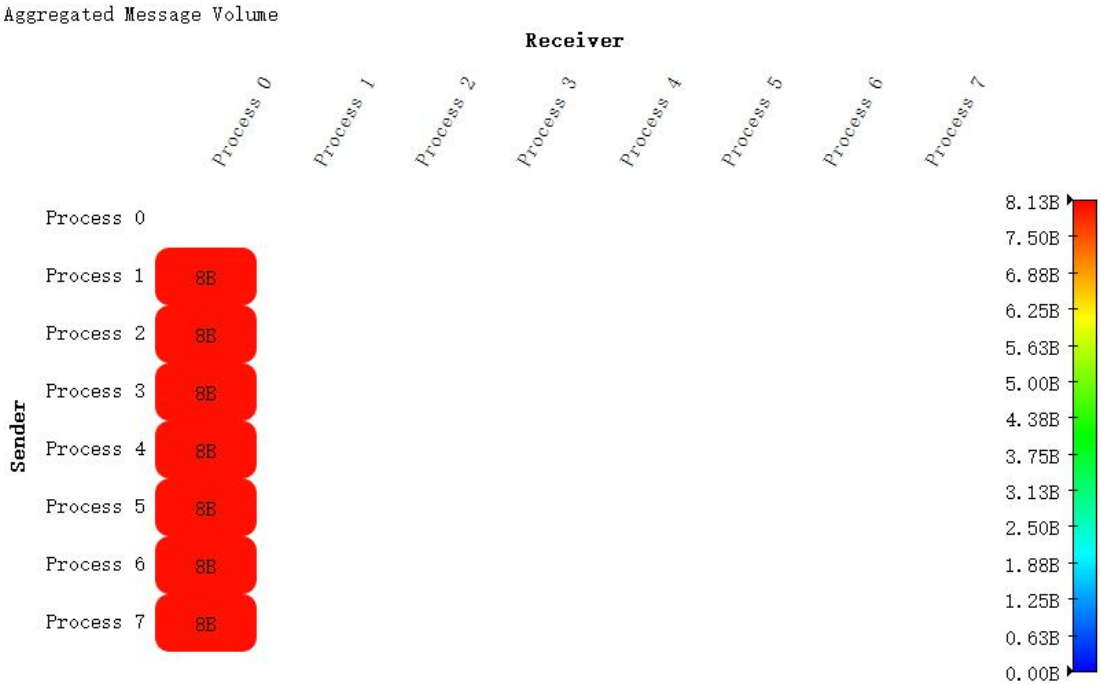
Basically, there are two part are involved with process communication: maxPieces info exchanged and output result synchronization.

1) Max Pieces info exchanged

The progress is shown as blow:

Figure 2.B.1

Process 0 will receive all the data from other processes first then calculate the global max piece(in the main part shown above). Then process 0 broadcast the global max piece out to other process. The aggregate message volume matrix is shown blow:



2) Output result synchronization

Once every process received the number of global max piece, all processes except process 0 go into a receiving blocking status to waiting for output the final result. The progress is shown below:

Figure 2.B.3

Once process n finish output result, it will send a message to the next process(process rank + 1) to tell it to process output result. From the graph, it's obviously to see there is a certain order from process 0 to 8.

Below the aggregate message volume matrix also shows that:

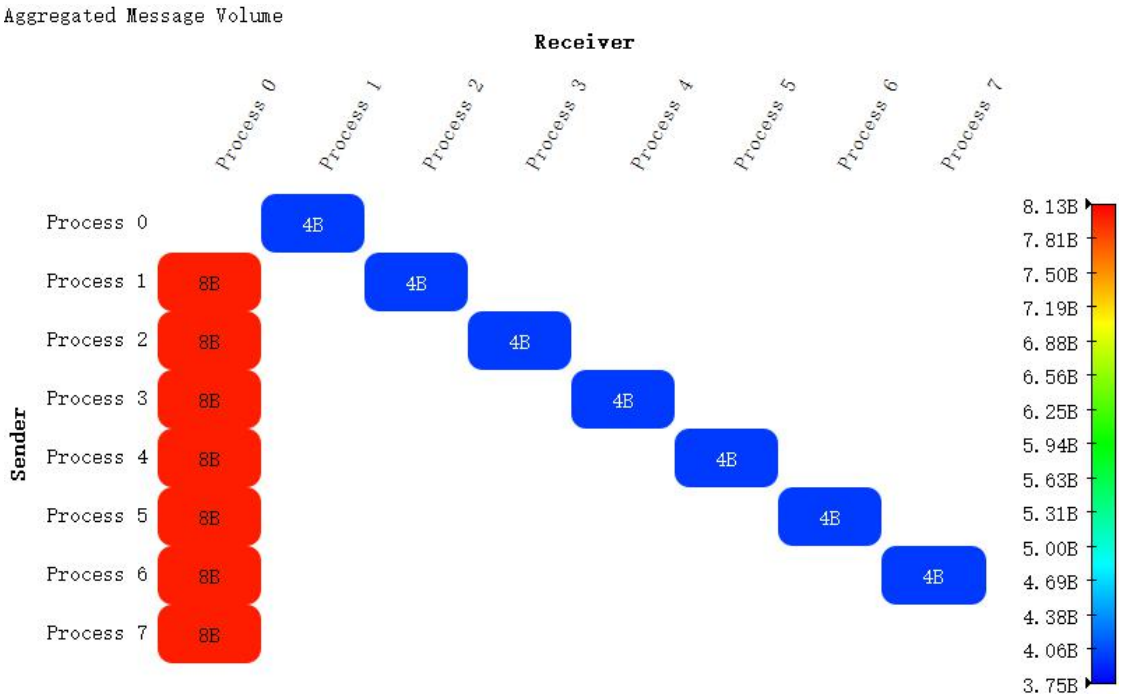


Figure 2.B.4

C) Data table

		N		
		3	4	5
Core	k			
64	8	0.027976	0.042788	8.460083
64	7	0.028206	0.039228	8.287172
64	6	0.040538	0.029844	8.709059
64	5	0.0278	0.02794	8.827954
64	4	0.02098	0.03283	8.620169
64	3	0.027927	0.045002	8.97512
32	8	0.012329	0.025486	4.847822
32	7	0.013027	0.017461	4.801054
32	6	0.013284	0.018141	4.445944
32	5	0.027643	0.008256	5.115601
32	4	0.012407	0.027978	5.116904
32	3	0.012219	0.03734	5.85012
16	8	0.001531	0.006431	2.897148
16	7	0.001073	0.005912	2.646419
16	6	0.000984	0.005919	2.878024
16	5	0.002402	0.006463	3.126778
16	4	0.000198	0.006208	3.050881
16	3	0.001018	0.007706	3.8184
8	8	0.001463	0.00697	2.075855
8	7	0.00068	0.007203	2.061816
8	6	0.000746	0.007135	2.099064
8	5	0.000134	0.006856	2.100095

8	4	0. 000161	0. 007206	2. 289258
8	3	0. 000169	0. 010111	2. 857559
4	8	0. 000161	0. 005452	2. 627709
4	7	0. 000102	0. 004599	2. 64442
4	6	0. 000175	0. 004526	2. 647974
4	5	0. 00017	0. 004561	2. 626826
4	4	0. 000108	0. 01057	2. 989276
4	3	0. 000165	0. 007659	4. 056746
2	8	0. 000138	0. 007222	3. 949529
2	7	0. 000126	0. 007154	3. 974982
2	6	0. 000143	0. 007175	3. 975882
2	5	0. 00013	0. 007198	3. 95856
2	4	0. 000135	0. 007814	4. 59256
2	3	0. 000242	0. 013027	6. 689173

Table 2.C.1

D) Time speed graph

N=3:

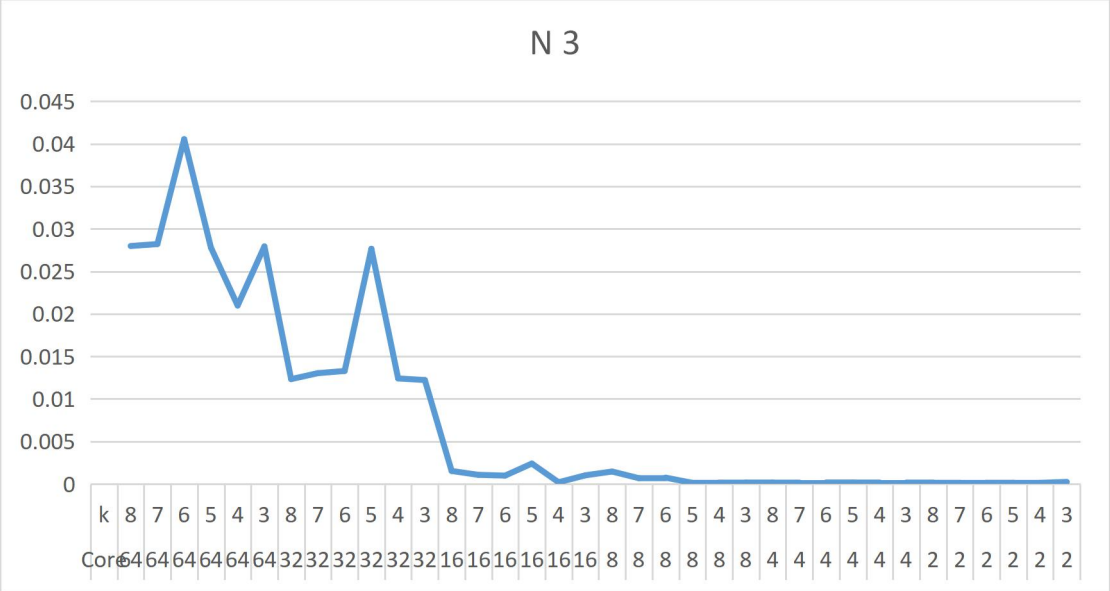


Figure 2.D.1

N=4:

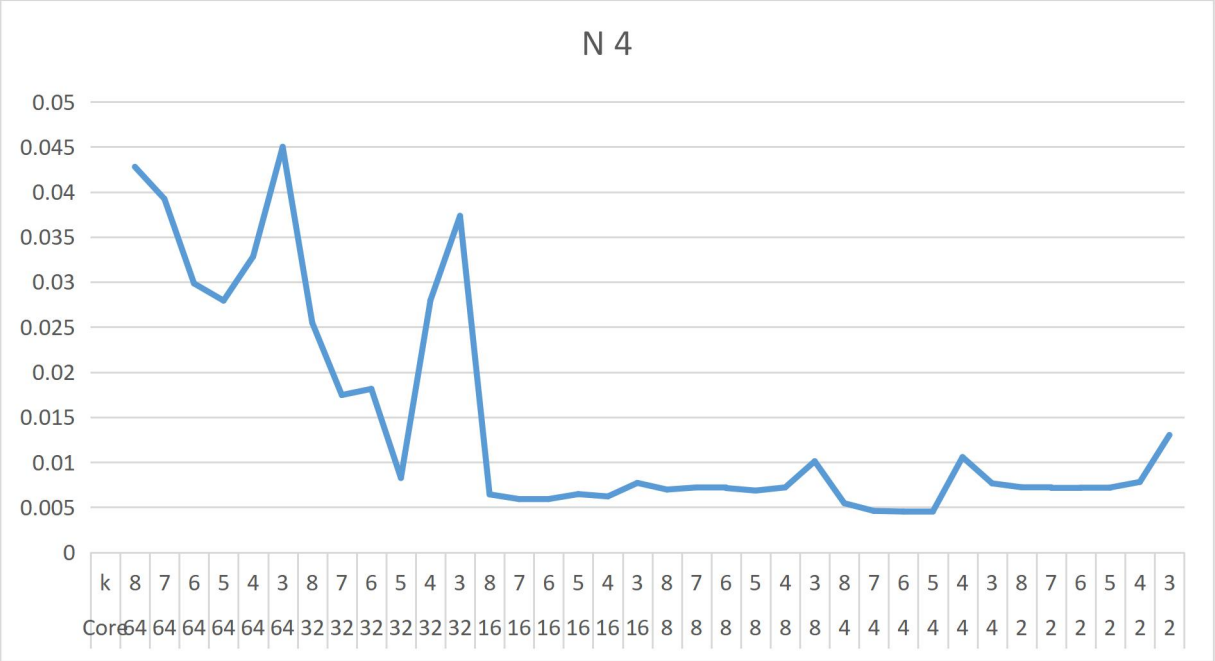


Figure 2.D.2

N=5:

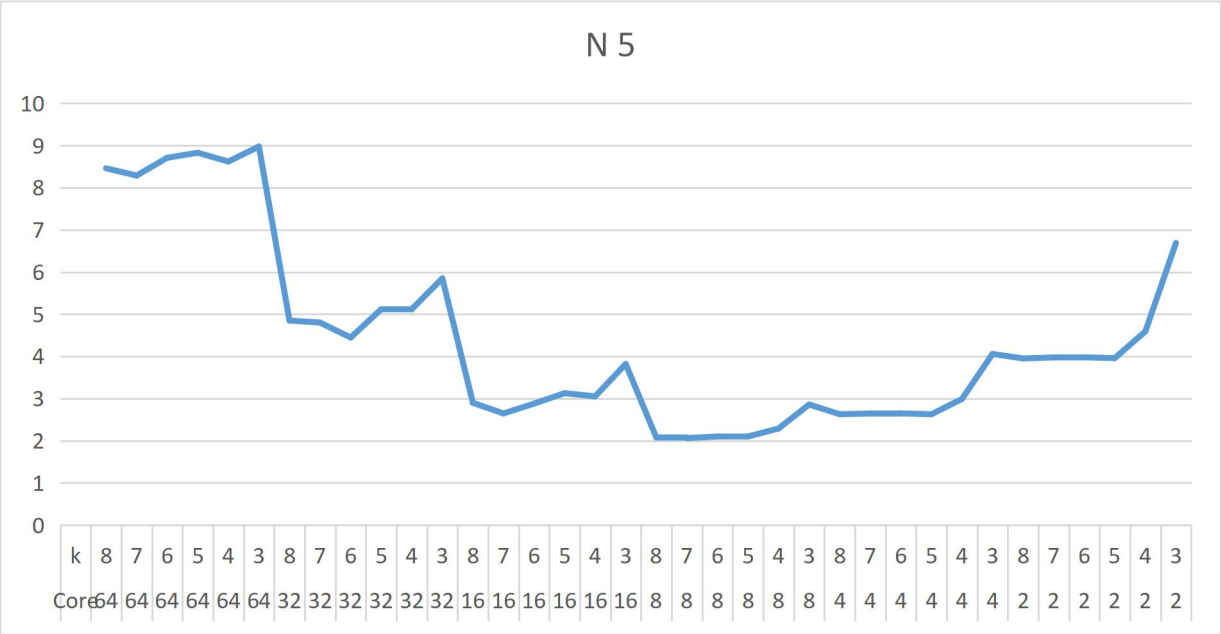


Figure 2.D.3

3. Limitation Discussion and Further Optimization

Currently, the parallel code is not so efficient. By conducting analysis and inspecting the code, the time cost mostly is on the part of adding chess pieces to the board. According to the result of experiment analysis, based on N=5,k=3,l=1,w=0 with 8 processes, the time cost on checking each board is only about 2s. But the time cost for exhaust the chessboard takes around 6s. In the algorithm implemented, each process will do the exhausting of chessboard, but it will only doing checking for every -np times(-np is the number of process). There is plenty of time wasted on just adding chess to the chessboard for each process but did nothing on checking. This is the part could be further improved.