Parallelized approach to the N-Queens Problem in Three Dimensions

Hong

June 7, 2022

1 Introduction

The original eight queens problem was proposed by Max Bezzel[2] in 1884 in a chess magazine. Frank Nauck extended the puzzle to N-Queens problem by increasing the number of queens and the size of the chessboard. The problem is to determine the placement of N Queens on a N × N chessboard such that no two queens can attack each other (A Queen can attack vertically, horizontally or diagonally). Fig.1 shows a solution when $N = 5$. The N-Queens problem has many real-world applications such as traffic control, circuit design, computer task scheduling, etc[1].

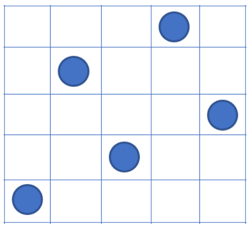
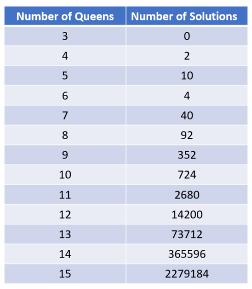


Figure 1: A solution to the 5-Queens problem

Obtaining all solutions for the N-Queens problem was proven to be NP-Hard, that is, there is no polynomial time algorithm exist. A common algorithm for obtaining all solutions is backtracking method, which incrementally build candidate solution and abandon(backtrack) the candidate when it is found invalid. Fig.2 shows the number of all solutions corresponding to number of queens found using backtracking methods.



Number of Queens	Number of Solutions
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712
14	365596
15	2279184

Figure 2: Number of two dimensional N-Queens solutions

As the size of the search space for all N-Queens solution is N!, for large N value, the time for running backtracking method is too long to be applicable. Thus, parallel algorithm and libraries have been utilized to speedup the solution finding process. For instance, Kise et al[4] use MPI on CPU to obtain all valid solutions for N = 24.

The N-Queens problem can be further extend to higher dimension[5] by defining how queens can attack in the additional dimensions. Chakiat et al[3] proposed the double backtracking algorithm to obtain a single solution for for N-Queens problem in three dimensions, where the chessboards are stacked up vertically and the queen can attack vertically in the new dimension (Fig.3). The algorithm can be easily modified to find all solutions, but is extremely computational intensive even with small value of N. The main purpose of this project is to improve and parallelize the double backtracking algorithm in order to obtain all the solutions more efficiently and quickly.

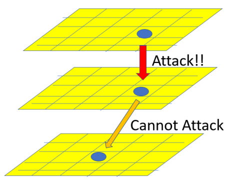


Figure 3: Illustrate attack position of N-Queens problems in three dimensions

2 Problem Statement

A set Γ with N element represents a solution of the three dimensional N-Queens problem if, for $i \neq j$, each element $<< x_i, y_i, z_i >, < x_j, y_j, z_j >>$ in Γ satisfies all of the following conditions:

For $z_i = z_j$ (if on the same chessboard):

(a) $x_i \neq x_j$ (not on same column)

(b) $y_i \neq y_j$ (not on same row)

(c) $x_i + y_i \neq x_j + y_j$ (not on same diagonal)

(d) $x_i - y_i \neq x_j - y_j$ (not on same diagonal)

For $z_i \neq z_j$ (if not on the same chessboard):

(e) $(x_i \neq x_j) \vee (y_i \neq y_j)$

For a specific N, the goal is to find all solution that satisfy the above conditions.

Notice that a two dimensional N-Queens solution can be represented as a one dimensional array A_N, where for i from 1 to N, a queen is located at (i, A[i]). Thus, a three dimensional solution can be represent as a matrix $M_{N\times N}$, as illustrated in Fig.4. In the following discussion, we will mainly use this kind of representation.

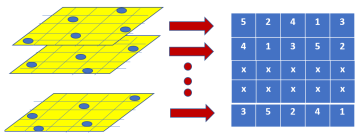


Figure 4: Converting three dimensional solution to a matrix.

3 Double Backtracking Algorithm

The main procedure of the double backtracking algorithm is described briefly as follows:

First, all 2D solutions for N are obtained using backtracking and store in a solution stack. 3D solution are then built from the solutions stack. The algorithm pick and add a element from the stack to the partial 3D solution. If the resulting partial 3D solution is valid, the algorithm will pick and add another element from the stack. If the resulting partial 3D solution is not valid, the algorithm will perform backtracking and remove the newest added solution and pick the next element relative to the removed element. When the 3D solution have height N and is valid, the solution is stored and backtracking is performed. When the algorithm end, all 3D solutions are found.

Fig.5 depicts the overall structure of the algorithm and Fig.6 illustrates how the backtracking step is performed.

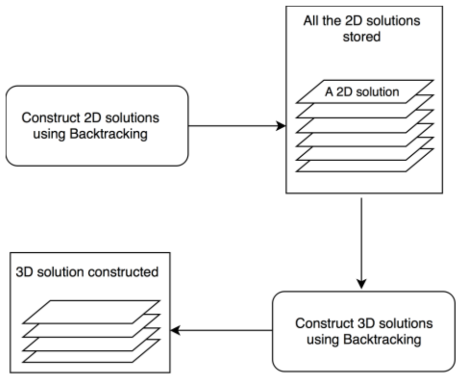

All the 2D solutions stored
A 2D solution
Construct 2D solutions using Backtracking
3D solution constructed
Construct 3D solutions using Backtracking

Figure 5: Abstract model [3]

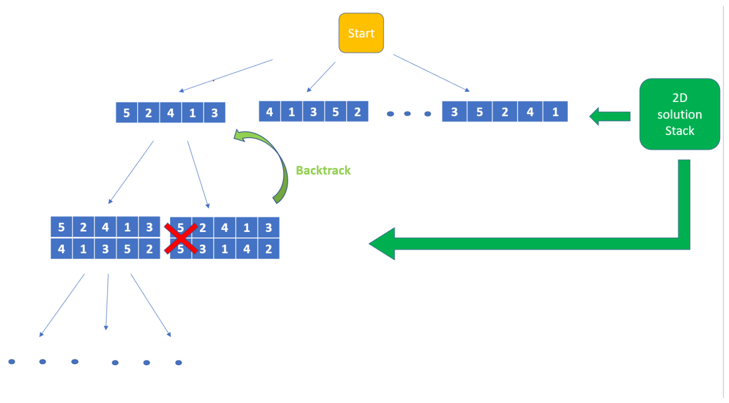
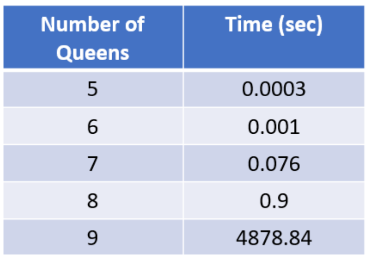


Figure 6: Illustration of backtracking steps in the algorithm.

4 Analysis of Double Backtracking Algorithm

Let S_N denote the number of solution for 2D N-Queens problem. The search space for the double backtracking algorithm is $\frac{S_N!}{(S_N-N)!}$, which grow extremely fast as N increases. For example, from Fig. 2, we can see that $S_9 = 352$, so the search space is $\frac{352!}{343!}$, which is approximately 8×10^{22}. When N = 10, the number go up to 4×10^{28} and even 5×10^{37} for N = 11.

To get a better feeling of how fast the computation demand grow, we run the algorithm on a single i3-4160 CPU, which has a maximum clock speed of 3.6 GHz. The result is shown in Fig 7. We can see that the computation time suddenly grow from less than one second to about one and a half hour when N go to 9. As the search space for N = 10 is a few order of magnitude larger than that of N = 9, it is not possible to obtain the result in any reasonable amount of time. In following sections, we will discussion how we speed up the running time of the algorithm.



Number of Queens	Time (sec)
5	0.0003
6	0.001
7	0.076
8	0.9
9	4878.84

Figure 7: The computation time of double backtracking algorithm on single CPU

5 Improvement of the Algorithm

5.1 Key Observation

One main method to achieve speed up is based on the following observation:

If Γ is a solution of the three dimensional N-Queens problem, then any permutation of Γ along the vertical dimension will also be a solution.

This simple observation (shown in Fig.8) allows us to reduce the computation significantly, since we can search in a certain subset instead of the whole search space. We can then perform permutation on the found solutions to obtain all the solutions.

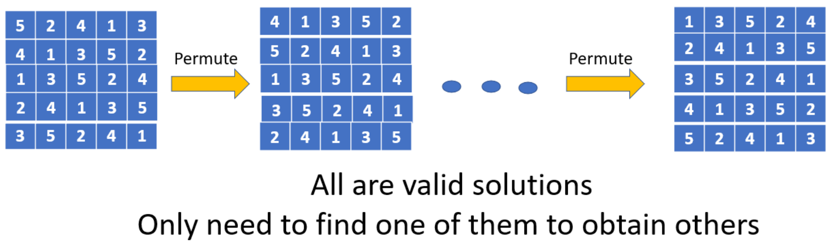


Figure 8: Key Observation

5.2 Proposed Algorithm

First, all 2D solutions for N are obtained using backtracking. Those solutions are then put into N solution stacks, stack1, stack2, ..., stackN, according to the value of the first entry of each solution (the row index where the queen located in the first column).

Main step:

3D solution are then built from those solutions stacks. When the partial 3D solution has height i, the algorithm will pick a solution from stack(i+1) and add it to the partial 3D solution. If the resulting solution is valid, it then go on the pick a solution from stack(i+2). If the resulting solution is not valid, then the algorithm backtrack and remove the newest added solution and pick the next solution from stack(i+1). When the 3D solution have height N and is valid, the solution is stored and backtracking is performed. When the algorithm end, all 3D solutions with their first column being an increasing sequence from 1 to N are found.

Fig.9 illustrates the main idea of the algorithm. Fig.10 shows a solution found when N = 5.

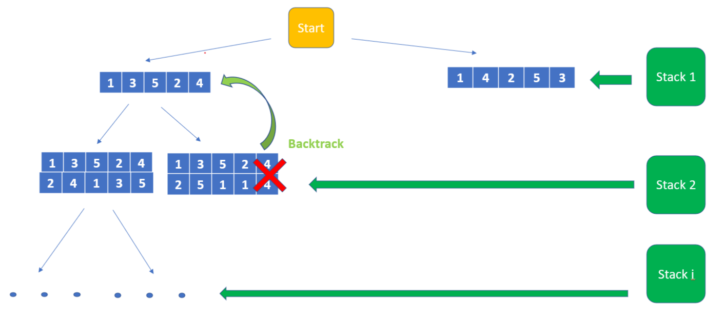


Figure 9: Illustration of backtracking steps in the proposed algorithm.

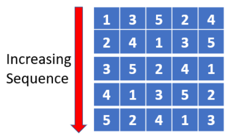


Figure 10: A solution found using the improved algorithm for N = 5

6 Analysis of the New Algorithm

We begin our analysis by inspecting the distribution of the 2D solutions corresponding to the value of their first entry. We test different value of N from 6 to 15 and obtain similar distribution. Fig. 11 shows the distribution for N = 11. We can observe that the 2D solutions spread across the y-axis with the same order of magnitude. Although the distribution is not uniform, to facilitate our analysis, we approximate the counts for each value using $\frac{S_N}{N}$.

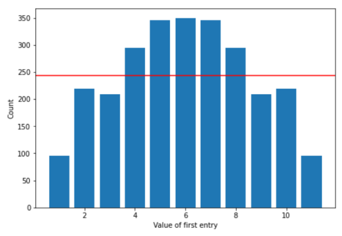
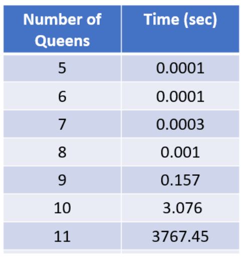


Figure 11: The distribution of number of solution with the same value of first entry for N = 11, the red line indicate the value of $\frac{S_{11}}{11}$

The approximate search space for our algorithm is $(\frac{S_N}{N})^N$ compare to $\frac{S_N!}{(S_N-N)!}$ of the original algorithm. After some simple calculation and approximation, our algorithm has reduced the search space for a factor of N^N, which is a significant improvement. To get a better feeling of the speed up, we run our algorithm on a single i3-4160 CPU again. From Fig.12, we can see that using the improved algorithm, less 0.2 second is needed for N = 9, which is much faster than 1.5 hour the original algorithm takes.



Number of Queens	Time (sec)
5	0.0001
6	0.0001
7	0.0003
8	0.001
9	0.157
10	3.076
11	3767.45

Figure 12: The computation time of our algorithm on single CPU

7 Parallelization

Even though significant speed up have been achieved using the new algorithm, we can see from Fig.12 that it still takes more than 1 hour to obtain all solutions for $N \geq 11$. Luckily, the purposed algorithm can be parallelized easily. In this section, we describe the parallelized algorithm and our implementation.

7.1 Parallelized algorithm

1. In parallel, obtain 2D solution stacks, stack1, stack2, ..., stackN, in the same fashion to the improved algorithm describe in section 5.2.
2. Divide stack1 to p segments, stack1_1, stack1_2, ..., stack1_p.
3. For i from 1 to p, run the main step of the purposed algorithm on stack1_i, stack2, ..., stackN in parallel.
4. Gather all the 3D solutions

7.2 Implementation

We use MPI and OpenMP for our parallelized algorithm using p processes with the following steps:

1. Launch p processes
2. Broadcast the number of queens, N, to all processes.
3. Each process calculates the 2D solutions.
4. Each process calculate the segment of stack1 it takes charge of using its rank value.
5. Each process then further divide the segment into t sub-segment
6. Each process launches t threads with each thread running the main step of the proposed algorithm on a sub-segment to obtain the 3D solutions (the 3D solutions' update is done in a critical region to prevent data race).
7. Each process send the count of solutions to the root process to facilitate the next step.
8. A gatherv collective communication is done to gather all 3D solutions to the root process.

7.3 Analysis

Let T_N^{2D} be the time used for solving the 2D N-Queens problem using serial algorithm, T_N^{3D} be the running time for the main step of the purposed serial algorithm for solving the 3D N-Queens problem, α be the latency and β be the time to send one unit of data. We will only consider the communication and computation demanding steps in the algorithm for analysis.

Step 2 takes: $(\alpha+\beta)\log_2 p$

Step 3 takes: T_N^{2D}

Step 6 takes: $\frac{T_N^{3D}}{pt}$

In worst case, step 7 takes: $p(\alpha+\beta)$

Step 8 takes around: $\alpha\log_2 p+\beta(p-1)N^2k$, where k is the average number of solution obtained by each processes.

Thus, the total time used for this parallelized algorithm is as follow:

$$T_{parallel}=(\alpha+\beta)\log_2 p+T_N^{2D}+\frac{T_N^{3D}}{pt}+p(\alpha+\beta)+\alpha\log_2 p+\beta(p-1)N^2k$$

Compare with

$$T_{serial}=T_N^{2D}+T_N^{3D}$$

7.4 Experiment

We conduct our experiment with a CPU cluster. Each node equipped with a i3-4160 CPU with 2 cores and each core can run two threads simultaneously using hyper-threading. We set N = 11 for our experiment.

We first test t = 1 and two processes each node to utilize the two CPU cores. From the scaling plot shown in Fig.13, we can see that it seems to achieve a linear speedup. Yet, we can see that for specific value of p (for example, when p = 14), the speedup is not as fast as anticipated, which we will tackle later.

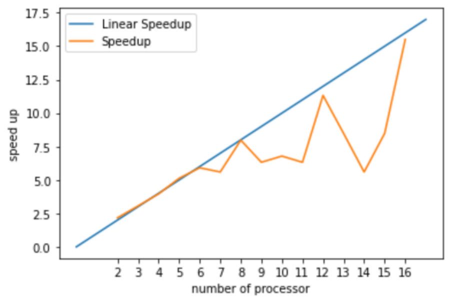


Figure 13: The speedup obtained with difference number of processors, for N = 11

Next, since each core can run two threads, we set different value of t to fully utilize the computational resources. Fig.14 shows the scaling plot for different value of t when p = 8. From the plot we can see that using two threads can double the speed compare to single-threading, but no significant speedup is observed if we further increase the value of t.

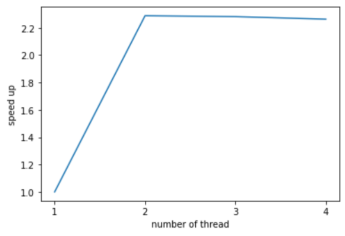
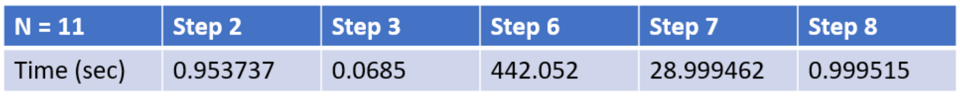


Figure 14: The speedup obtained with difference number of threads with 8 processors, for N = 11

We also test whether the number of process in each node will affect the performance (We can only launch two processes in each node due to some compute resource limit), but no noticeable different are found. For example, running 16 processes on 16 nodes takes 242 seconds, while on 8 nodes takes 244 seconds.

Last, we also test the time each step we consider in section 7.2 takes. The result is shown in Fig.15. We can see that Step 6 accounts for majority of the time consumed.



N = 11	Step 2	Step 3	Step 6	Step 7	Step 8
Time (sec)	0.953737	0.0685	442.052	28.999462	0.999515

Figure 15: The time for each step using N = 11, p = 8, t = 1. Step 2 is the broadcast operation. Step 3 is calculating the 2D solutions. Step 6 is the calculation of 3D solutions. Step 7 is the communication for number of solution each process obtained. Step 8 is the gatherv operation.

8 Discussion

From Fig.13, we can see that linear speedup is achieved at least for p = 16, which is reasonable since from Fig.15, we can see that the communication overhead and the time for calculating 2D solutions takes much less time compared to the parallelized region(Step 6).

One question left is why, as shown in Fig.13, for some value of p, we do not observe the expected speedup. After carefully inspecting the algorithm and data, we found that the problem comes from the imbalance workload caused by some implementation issue while performing 2D solution stack distribution (Step 4 in section 7.2). In the original implementation, when the number of 2D solution in stack1 is not divisible by the number of processes, the algorithm distribute the remainder to the root process. This strategy is fine for small number of processes, but when the number of process increase it becomes a issue since the root process will takes too many tasks compare to other processes. For example, as the size of stack1 for N = 11 is 96, the root processes will be assigned 18 tasks, while others only 6. This explain the sudden drop of performance for some value of p in Fig.13.

After spotting this issue, we adjust our algorithm so that the remainder will be distributed evenly to each process. Then we run the experiment again. From Fig.16, we can see that the performance is much better after the adjustment.

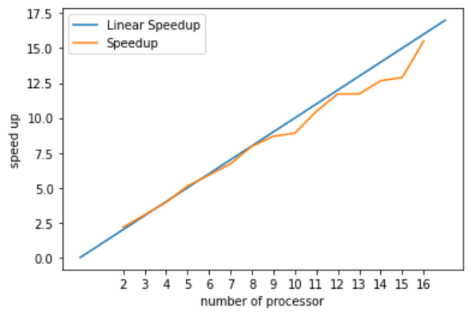


Figure 16: The speedup obtained with difference number of processors after copping with the load imbalanced issue, for N = 11

9 Minor improvement

One minor improvement to the algorithm is to further parallelize the step for obtaining the 2D solutions using OpenMP or MPI (By a factor of 2-3 for small N value) . But from Fig.15, we can see that the step(Step 3) only accounts for very little proportion of the computation time. Thus, the improvement is marginal.

10 Conclusion

In this project, we extend the original double backtracking algorithm for getting a single solutions for the 3D N-Queens to obtaining all solutions. We then improve the algorithm by one key observation that the any permutation of a single solution is also a valid solution. Next, we further speed up the algorithm by parallelization using MPI and OpneMP. Last, we conduct analysis and experiment which allows us to spot and solve the load imbalanced issue for further improvement.

References

[1] J. Bell and B. Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1–31, 2009.

[2] M. Bezzel. Proposal of 8-queens problem. *Berliner Schachzeitung*, 3(363):1848, 1848.

[3] A. Chakiat, A. Sudhakaran, A. A., and P. Venkatesh. A novel double backtracking approach to the n-queens problem in three dimensions. *International Journal of Computer Applications*, 169:1–5, 07 2017.

[4] K. Kise, T. Katagiri, H. Honda, and T. Yuba. Solving the 24-queens problem using mpi on a pc cluster. *Graduate School of Information Systems, The University of Electro-Communications, Tech. Rep*, 2004.

[5] S. P. Nudelman. The modular n-queens problem in higher dimensions. *Discret. Math.*, 146:159–167, 1995.