Parallel Implementation for N Queens Problem

Tianyi Mo
The University of Melbourne
tianyim@student.unimelb.edu.au

ABSTRACT

The n queens problem is a generalization of the classic 8 queens problem. The problem has time grows exponentially with the growth of queens thus is classified as NP (Non-Deterministic Polynomial) problem[3]. Due to the time complexity of the problem, computation of the number of solutions is time-consuming. With the development of modern multi-core computers and cluster computer, the classic problem can be solved more efficiently. This report will introduce a parallel algorithm for n queens problem that accelerate the process of finding all solutions of n queens problem. It will also show the experimental results of and C++ implementation of the parallel algorithm and discuss the speedup and overhead of the algorithm.

KEYWORDS

N Queens Problem, Parallel Algorithms, MPI

1 INTRODUCTION

1.1 Problem

The n queens problem is originated from the chess, and has many applications in real-world problems such as choosing the positions for fire brigade, shopping malls or hospitals [9]. The problem is defined as placing n queens on a $n * n$ chessboard and ensuring that there is only one queen on each row, column and diagonal. A normal solution any chessboard satisfied the constraints and a fundamental solution is got by treating the rotation or symmetry of a solution as the same solution. Even though the problem is simple to describe, there is no closed-form formula for the number of all or fundamental solutions. The table 1.1 shows some solutions in n queens puzzle, it can be seen that the number of solutions grows quickly with n.

n	fundamental	all
1	1	1
2	0	0
3	0	0
4	1	2
5	2	10
6	1	4
7	6	40
8	12	92
9	46	352
10	92	724
11	341	2,680
12	1,787	14,200

1.2 Related works

The brute force method is the simplest method for counting the number of solutions is to generate all possible combinations of the queens with the constraint that there is only one queen on each column, then check whether it is a valid solution. This method is extremely inefficient because it has to try all n^n combinations.

Another method to find a solution is the minimum-conflicts heuristic [4], unlike the previous method placing one queen every time, it starts with all randomly placing one queen on each column. In each step, the queen is moved to locations with the minimum number of conflicts. This heuristic-based algorithm can find a solution for large n queens problems in a short time, but it is infeasible to used to count the total number of solutions because it involves comparing a new solution to all previously found solutions. The comparison will cause $O(m^2n)$ steps to complete where m is the number of solutions. Coarse-grained parallel genetic algorithm(CPGA) and Ant Colony Optimization(ACO) can be used to solve the n-queen problem but has similar drawbacks as minimum-conflicts heuristic in solution finding all solutions [3][9].

1.3 Sequential Algorithm

The sequential algorithm used is an modified from the recursive algorithm described in [10] by adding counts of the total number of solutions [7].

In this algorithm, it firstly places a queen on the first column, then recursively tries to place a queen on next column. When a potential location is found, it checks whether the chessboard is valid after placing a queen on that location. The algorithm found a solution when there is a queen placed on the last column and does not violate any constraints, then it increases the counter. **Algorithm 1** shows the pseudo-code for the sequential algorithm, it recursively calls its self until reaches the rightmost column or cannot find a valid solution.

Algorithm 1: Sequential algorithm

```
recursive-PlaceQueen(n, chessboard, column)
if column < n then
    update solutionCount
    return True
end
for row ← 0 to n do
    if validChessBoard (n, chessboard, row, column) then
        put queen on (row, col)
        recursive-PlaceQueen(n, chessboard, column + 1)
    end
end
return False
```

Despite rotation and symmetry, this algorithm is guaranteed to find unique solutions and therefore does not have to store all initially found valid solutions. Besides, unlike other algorithms that has a random start state, its algorithm does not have any randomness thus gives more stable running time. However, it has

time complexity $O(n!)$ and does it in parallel could reduce the time for finding the solution.

2 PARALLEL ALGORITHM

The task of counting the number of solutions can be divide into several sub-tasks. The parallel algorithm is based on firstly generating several sub-tasks, then distribute them on a number of processes and collect the results from processes.

2.1 Sub-tasks

In the sequential algorithm, each step it try to place a queen on each location of that column. The sub-tasks are actually chessboard that few columns are already filled with queens and other columns are empty. In the n queens problem, if k first columns are filled with queens, it will produce n^k sub-tasks before checking for validity. For example, if the $n = 10$ an and $k = 2$, there are 100 sub-tasks where 72 of them are valid. Since the k must be an integer, the number of sub-tasks can only be $k^1, k^2, ..., k^n$. All these sub-tasks can be handled independently and results sum to the total result.

In order to minimize the total execution time, sub-tasks should have an appropriate size. If k size is too large, it could result in an imbalance of workload on each process. For example, when k is equal to 1, there are only n sub-tasks and it can easily have more processes than number of sub-tasks. However, a large k size could introduce frequent communication between the root process other processes and introduce significant communication. In addition, a large k size also costs more time to generates sub-tasks because the generating step is sequential. Therefore, 2, 3 or 4 might be reasonable values for k.

2.2 Dynamic Assignment

The workload balancing problem is the minimization of the difference between the workload of the bottleneck node and the workload of the fastest node [6]. In this task, each sub-task does not have a similar execution time because some of them will be pruned during the search and end earlier than other sub-tasks.

Thus it is important to find a mechanism to balance the workloads. The static and dynamic assignments are two common strategies to distribute works processes. The static assignment has less communication overhead but requires accurate estimation on execution time while the dynamic assignment is better when the execution time is unknown [5]. In the parallel n queens problem, it is not easy to estimate the execution time of each sub-task. A globally shared work queue managed by the root process is used to balance the workload [1]. The root is responsible for finding the next unsolved sub-task and transmitting to the process when the previous sub-task is completed.

2.3 Communication

The messages from in the root process to other processes include sending the sub-task id and sending the stop signal to inform them that there are no more tasks to do. The messages from other processes to the root process are to inform that the current sub-task has been finished. Rather than transmitting to the root node when each task is finished, the results are accumulated in each process and reduce to the root node at the end to make the communication more efficient.

3 EXPERIMENTS

3.1 Implementation

Both sequential and parallel algorithms are implemented in C++ and the parallel version uses Message Passing Interface (MPI). The MPI requires explicit communication between processes [2]. This is because each sub-task will not produce a significant amount of data which is used later in other sub-tasks. Thus, each sub-task can be done efficiently without a shared memory model and OpenMP. In the implementation, for x available cores, one core (rank0 process) is used for scheduling and all $x - 1$ cores are used for calculation.

3.2 Performance of Using Multiple Nodes

Since there is a limited number of CPUs on each node of the cluster, each node has 32 cores on snowy partition [8]. If we want to run the experiment on more than 32 CPUs cores, we must use more than one node. Using multiple nodes may slow down the execution because of cluster topology and it is crucial to verify whether running on different nodes will affect the performance. Firstly the 16 queens problem is running on 2 to 12 process where all process is located on a single cluster node. Then the same problem is running on 2 to 12 process where each process is located on different nodes of the cluster. The **Figure 1** shows that their performance is extremely close thus multiple nodes can be used without affecting performance.

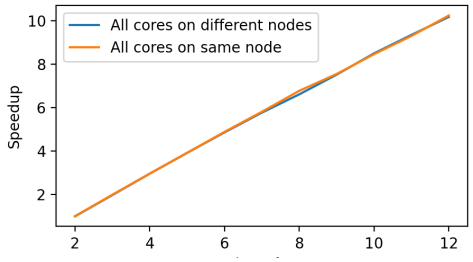

Figure 1: Comparison of using single and multiple nodes

3.3 Communication Overhead

The experiments are run on the snowy partition of the spartan cluster. It cost the sequential algorithm 132.388 seconds to get the answer for 16 queens problem 14772512. In contrast, the 2 core parallel algorithm (only 1 core actually for computation) costs 134.4 seconds. Doing these experiments repeatedly gives similar results because there is no randomness in this algorithm. It can be seen that there is only 2.012 seconds (or 1.51 %) difference between these two experiments.

3.4 Speedup

The parallel implementation of the algorithm has significantly reduced the execution time from 134.4 seconds to 3.674 seconds. **Figure 2** shows the speedup of 16 queens problem. The speedup

ratio is calculated by dividing the sequential time. Despite some outliers, the increase in the number of cores will result in a reduction in execution time and an increase in speedup. It can be seen that the speedup is close to linear when the number of cores is less than 15. For example, the speedup is 10.16 when using 12 cores. However, when the number of cores greater than 15, the speedup is still increasing but at a slower speed. For example, the speedup is 36 when using 64 cores, this might because these not enough sub-tasks for a large number of processes to have workload balanced. **Figure 3** shows the relationship between the number of cores and the Number of cores ∗ Execution time. It illustrates that as the number of cores, the overhead also increases and efficiency decreases.

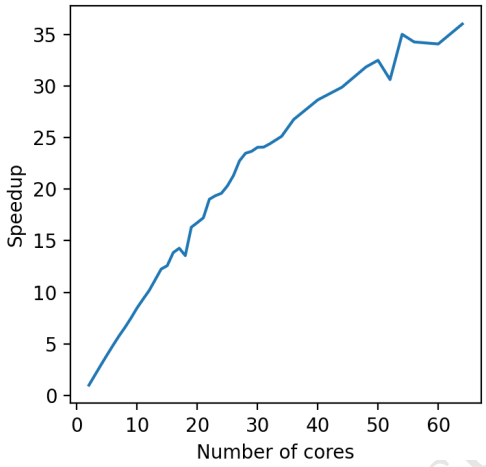

Figure 2: Speedup of parallel algorithm

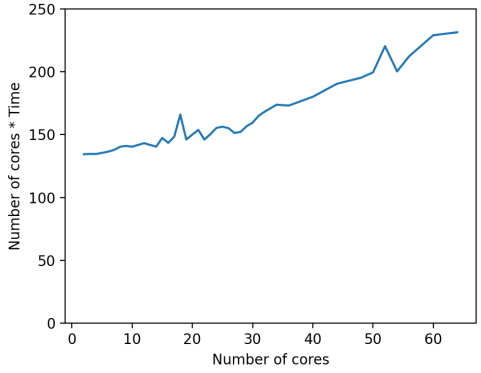

Figure 3: Total work for different number of cores

4 CONCLUSION

In conclusion, this paper analysis some popular algorithms for n queens problem and discuss the pros and cons of paralleling them. Then it chooses the algorithm that can take the benefits of multiple cores on the cluster computers. The parallel algorithm consisting of splitting the tasks and assigning dynamically to the processes. From the results, it is interesting to found that using CPUs on multiple nodes does not affect the performance. The experiment results show that the parallel algorithm can significantly reduce the execution time of finding all solutions for n queens problem. However, when the number of cores increase, the efficiency of the parallel decrease. Using less than 15 cores could be reasonable choice as the accelerate the algorithm while keeping high efficiency at the same time.

The splitting of the task could be further investigated. The current approach could give n^k sub-tasks with different execution times. Since it is found that the number and size of sub-tasks could affect load-balancing, it could be better if can have a more flexible method that can find sub-tasks with predictable execution time.

REFERENCES

[1] J. Dinan et al. "Dynamic Load Balancing of Unbalanced Computations Using Message Passing". In: *2007 IEEE International Parallel and Distributed Processing Symposium*. 2007, pp. 1–8. DOI: 10.1109/IPDPS.2007.370581.

[2] Conveners Jack Dongarra David Walker and Meeting Chairs. *MPI: A Message-Passing Interface Standard Version 3.0*. Tech. rep. Message Passing Interface Forum, Sept. 2012. URL: https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.

[3] S. Khan et al. "Solution of n-Queen problem using ACO". In: *2009 IEEE 13th International Multitopic Conference*. 2009, pp. 1–5. DOI: 10.1109/INMIC.2009.5383157.

[4] Steven Minton et al. "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems". In: *ARTIF. INTELL* (1992), pp. 161–205.

[5] Todd Mowry and Brian Railing. *Parallel Computer Architecture and Programming*. URL: http://www.cs.cmu.edu/afs/cs/academic/class/15418-f20/public/lectures/05_progperf1.pdf. (accessed: 30.10.2020).

[6] Yassine Ouazene et al. "Workload balancing in identical parallel machine scheduling using a mathematical programming method". In: *International Journal of Computational Intelligence Systems* 7 (Supplement 1 2014), pp. 58–67. ISSN: 1875-6883. DOI: https://doi.org/10.1080/18756891.2013.853932. URL: https://doi.org/10.1080/18756891.2013.853932.

[7] Martin Richards. *Backtracking Algorithms in MCPL using Bit Patterns and Recursion*. 1997. URL: https://www.cl.cam.ac.uk/~mr10/backtrk.pdf.

[8] *Spartan Documentation*. URL: https://dashboard.hpc.unimelb.edu.au/status_specs/. (accessed: 20.10.2020).

[9] Kaiyu Wang, Zhenzhou Ji, and Yihao Zhou. "A Parallel Genetic Algorithm Based on MPI for N-Queen". In: 2017.

[10] Niklaus Wirth. *Algorithms + Data Structures = Programs*. 1st edition. Prentice-Hall, 1976. ISBN: 8120305698.