CS-301
N Queens problem

Deep Godhani
201701418

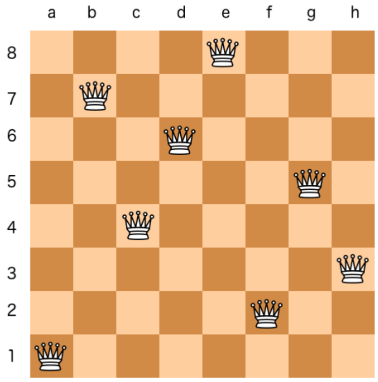
Sairam Venkatachalam
201701433

May 2, 2020

1 Introduction

The N queens problem is a classic chess puzzle where our objective is to place N queens on an N x N board such that no two queens threaten each other.This would mandate that no two queens share the same row, column, or diagonal.

The classic version of this puzzle uses 8 queens on the regular 8 x 8 chess board. One of its solutions is given below.

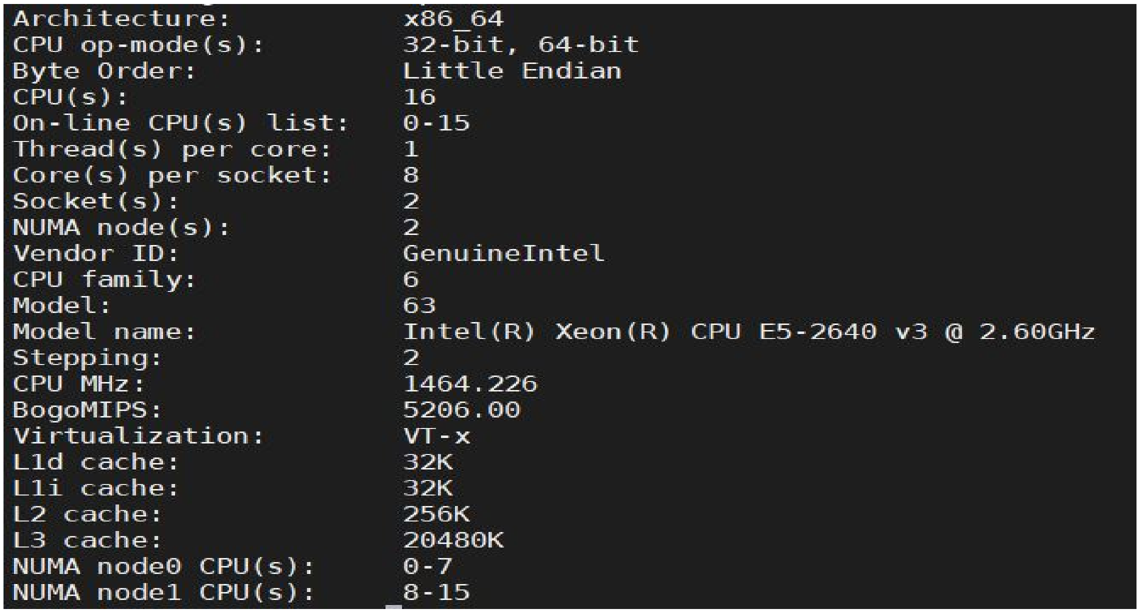


2 Applications

The analysis of chess positions by a chess engine is often done using a backtracking approach similar to that used in this problem.

This technique is can be useful in deadlock prevention during process execution. It is can also be used for memory management in disks.

3 Hardware details



```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    1
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
Stepping:              2
CPU MHz:               1464.226
BogoMIPS:              5206.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0-7
NUMA node1 CPU(s):     8-15
```

4 Serial Algorithm Approaches

4.1 Brute Force Approach

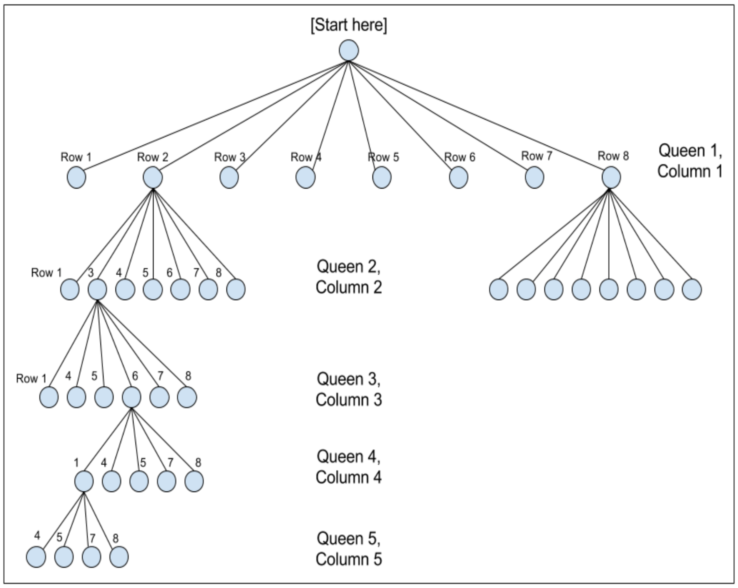
The first approach which comes to mind is the brute force approach where we would check all possible placements of N queens on an N x N board and check which placements result in the solution. This method would also check many fruitless positions and hence would take the longest execution time.

In particular for an N x N chess board there are N^2 squares to place N queens which results in N^2C_N different positions to analyze.

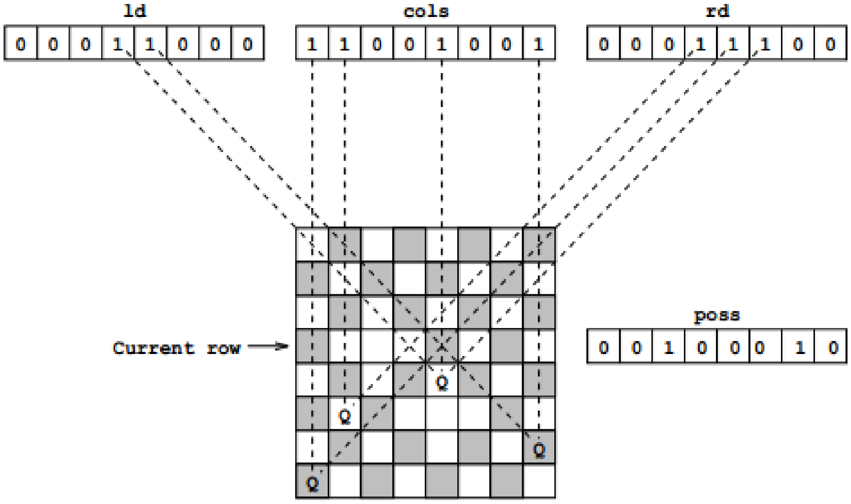
4.2 Backtracking

In this method we start by placing a queen on the first row and then move on to the next row placing queens in the columns where there is no conflict. At any point during this process if we are unable to place the queen in any of the columns for a particular row, we resort to backtracking to the previous row and the process repeats. This solution uses a recursive method in order to perform the back tracking.

The main advantage of this solution over the brute force method is that it avoids completely redundant positions. After knowing that a queen can not be placed on any columns for a certain row, we do not have to check for all the subsequent rows for that configuration as all of them are invalid solutions.



4.3 Bit Masking



In order for us to be able to place a queen on a given column of the current row, we must check the following 3 conditions.

1. There must not be be any other queen in the same column

2. There must not be be any other queen in the left diagonal of this cell.
3. There must not be be any other queen in the right diagonal of this cell.

In order to optimize this process, we can store this information in bits which we can use to determine whether the queen can be placed. We do this using bitmasking. In this process we maintain arrays known as bit maps for each of three bit arrays ie. column array and left and right diagonals array by replacing bit 1 in place of bit 0 to indicate the presence of queen at certain position.

5 Scope of Parallelization

In the case of a serial backtracking algorithm, we saw that the placement of the queen in the current row depends only on the previous row's queens position. For example, the placement of the queen in the second row depends on the position of the first row's queen. Similarly, the position of the third-row queen will depend on the position of the first and the second row's queens and so on .. continues for other rows.

Moreover, in the given serial algorithm, we saw that the backtracking algorithm has many independent paths since there exists more than one possibility of placing the queens in the single row.

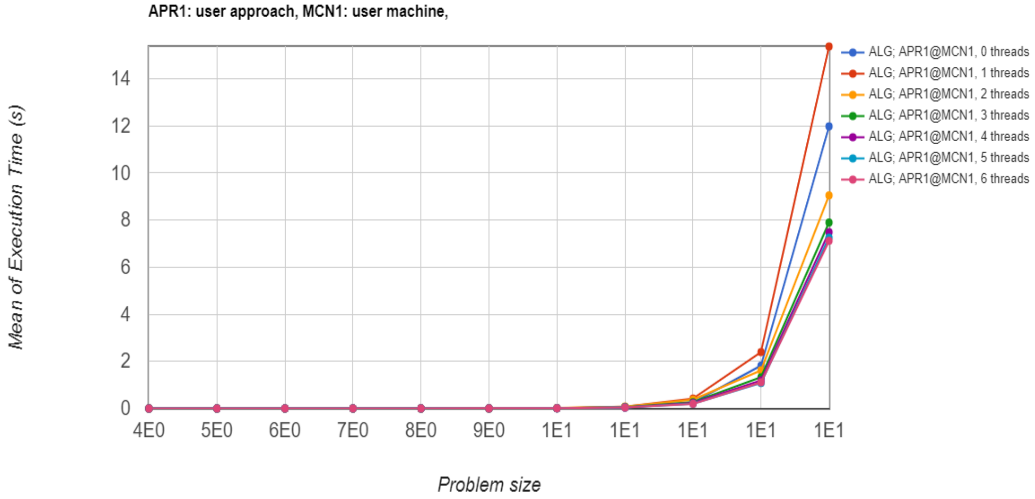
Taking this into consideration we have designed different parallelization techniques to overcome the large execution time and also, we take into consideration different problems like synchronization and load balancing.

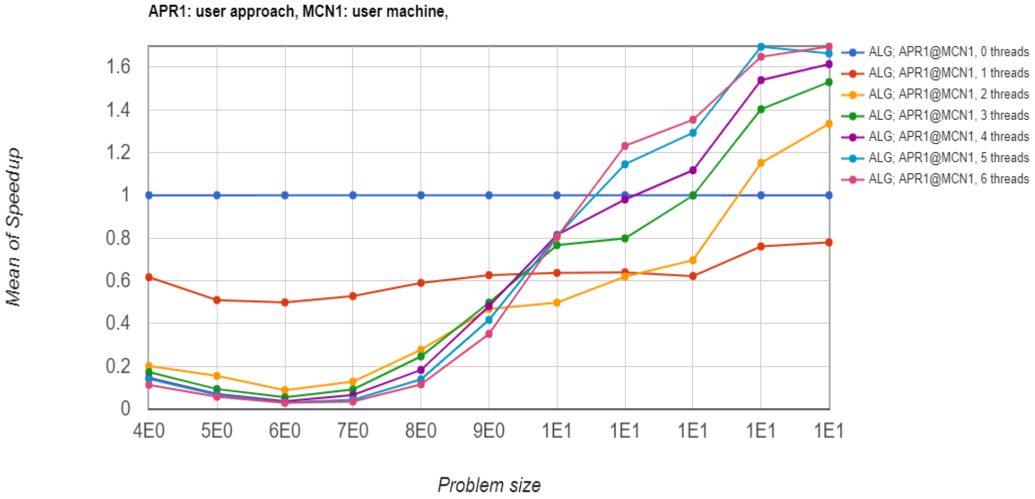
6 Parallelization strategies

During the process of parallelization, we find out that there are several solutions that are found independent of each other where the queens either start of at different positions on the first row or Branch out subsequently. Thus these processes can be performed simultaneously by distributing the load amongst different processors and simultaneously decrease the total execution time.

6.1 Method 1 : Parallelization of Backtracking Algorithm

In this method we divide the task between the processors by generating small tasks at each row of the N x N Board by using the backtracking algorithm. This tasks which are provided to the processors to execute are following the load balancing since all the tasks are same as a result the threads do not remain in idle state but in this method the task which are distributed are dependent on each other. Due to this task dependence the total time of execution increases.


APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
0
2
4
6
8
10
12
14
4E0
5E0
6E0
7E0
8E0
9E0
1E1
1E1
1E1
1E1
1E1
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads
ALG; APR1@MCN1, 5 threads
ALG; APR1@MCN1, 6 threads


APR1: user approach, MCN1: user machine,
Mean of Speedup
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
4E0
5E0
6E0
7E0
8E0
9E0
1E1
1E1
1E1
1E1
1E1
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads
ALG; APR1@MCN1, 5 threads
ALG; APR1@MCN1, 6 threads

Results-quantitative Analysis-discussion

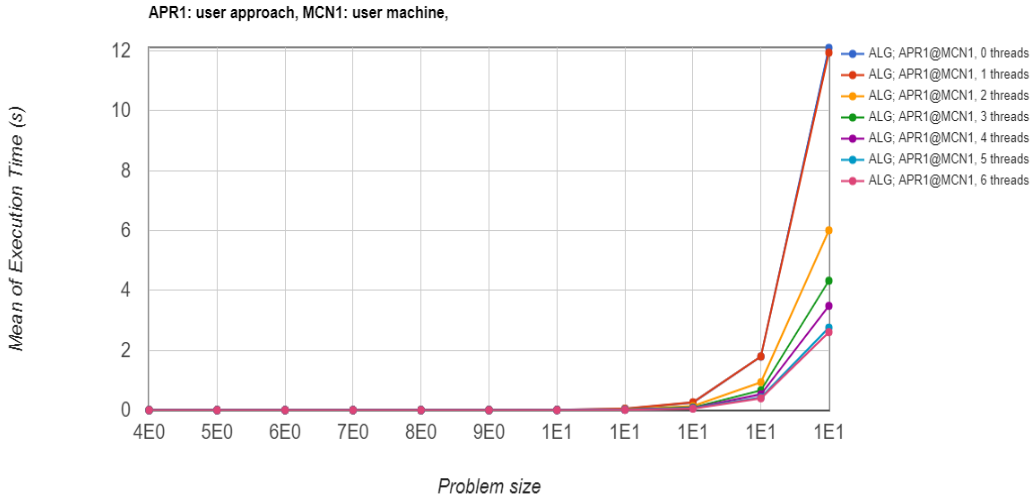
In the case of the brute force method it will look through every position on an N x N board, N times, for N queens.It is by far the slowest and most impractical method.The time complexity of the algorithm will be $O(n^n)$. Even if we prevent it from checking queens occupying the same row as each other, it will still be brute force, but the possible board states drop from 16,777,216 to a little over 40,000 for an 8 x 8 board and the algorithm has a time complexity of $O(n!)$. The Backtracking approach is a recursive approach which reduces the number of positions to be checked significantly.The time complexity in this case is found to be exponential estimated to be $O(2^n)$.

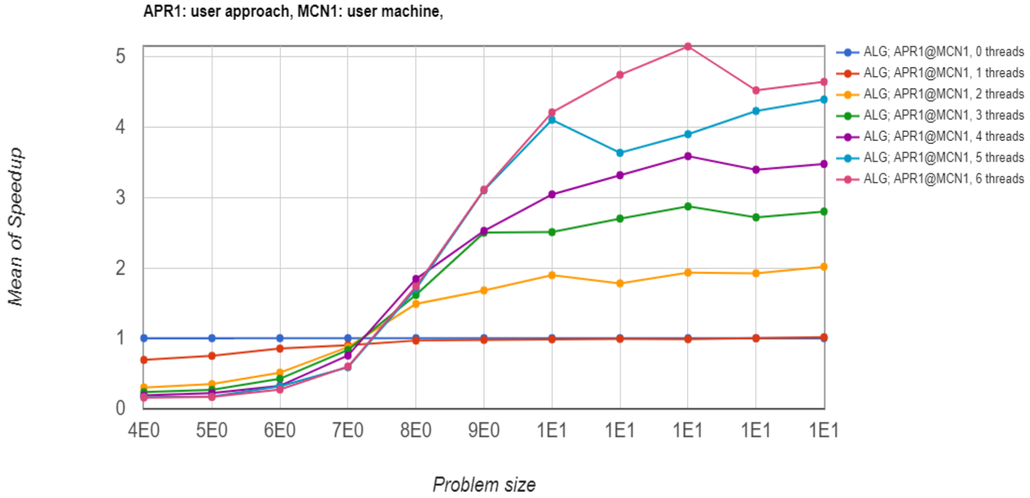
For this problem we use a single function in order to check the validity of the placement of a queen in a certain row. This function checks the diagonals and the column to make sure there is not conflict and in all for each existing path we do n accesses. Hence there are about $2n^2$ memory accesses in a single path. In total the brute force method checks for all possible positions and hence has a total of $n^2C_n * 2n^2$ memory accesses.The backtracking approach makes about $2^n * 2n^2$ memory accesses to check the validity.

The total time of execution goes on decreasing as the number of total processors are increased. But we observe that as we go on increasing no of processors the improvement in execution time goes on decreasing. Execution time is highest for method-1 because of presence of dependent tasks while execution. The speedup of the program increases with increase in number of processors. The efficiency of the algorithm goes on decreasing as the processors count increases.

6.2 Method 2 : Parallelization of Backtracking Algorithm

In this method we assign each column of the first-row to each processor to execute and find all the solutions with a certain placement of queen. In this method the work is not distributed equally because it is possible that a particular column has more solutions while some columns don't have even a single solution. But the advantage of this method is that the task generated are independent of each other so can be executed simultaneously.


APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
0
2
4
6
8
10
12
4E0
5E0
6E0
7E0
8E0
9E0
1E1
1E1
1E1
1E1
1E1
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads
ALG; APR1@MCN1, 5 threads
ALG; APR1@MCN1, 6 threads


APR1: user approach, MCN1: user machine,
Mean of Speedup
0
1
2
3
4
5
4E0
5E0
6E0
7E0
8E0
9E0
1E1
1E1
1E1
1E1
1E1
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads
ALG; APR1@MCN1, 5 threads
ALG; APR1@MCN1, 6 threads

Results-quantitative Analysis-discussion

The Backtracking approach is a recursive approach which reduces the number of positions to be checked significantly. The time complexity in this case is found to be exponential estimated to be $O(2^n)$. The space complexity of the above algorithm is $O(n)$ array to store the positions of previous row's queens.

The memory accesses done in this method are same as method-1 that is $2n^2$ accesses for each path in the grid, resulting in total of $2^n * 2n^2$ memory accesses.

The total time of execution goes on decreasing as the number of total processors are increased. But we observe that as we go on increasing no of processors the improvement in execution time goes on decreasing. Execution time is observed to be less in method-2 as compared with the method-1 because in this method the tasks are independent of each other but still its distribution is not equal. The speedup of the program increases with increase in number of processors. The speed up in this case is more as compared that of method-1 because of simultaneous execution of tasks. The efficiency of the algorithm goes on decreasing as the processors count increases.

6.3 Method 3 : Parallelization of Backtracking Algorithm

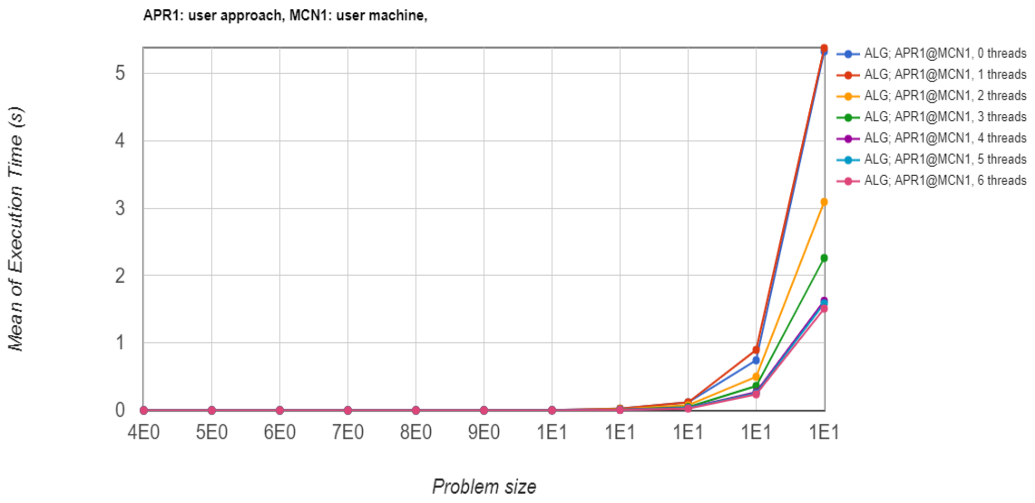
In this method we select a certain depth (rows) till which we perform the placement of queens level-wise will after which we search each solution linearly. This method leads to the optimization to the middle ground between the first-two methods of parallelization.

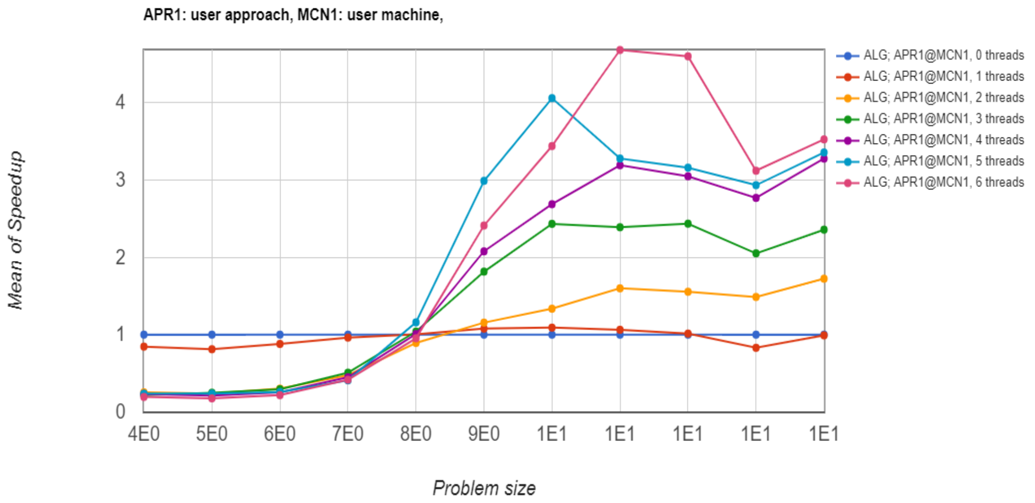
Results-quantitative Analysis-discussion

The Backtracking approach is a recursive approach which reduces the number of positions to be checked significantly.The time complexity in this case is found to be exponential estimated to be $O(2^n)$. The space complexity of this algorithm is $O(n + k) \approx O(n)$ where k is a constant, it also stores the array to remember the previously placed queens position along with that it stores few variables which are used to calculate the symmetry solutions in N x N board.

The memory accesses done in this method are same as method-1 that is $2n^2$ accesses for each path in the grid, resulting in total of $2^n * 2n^2$ memory accesses.

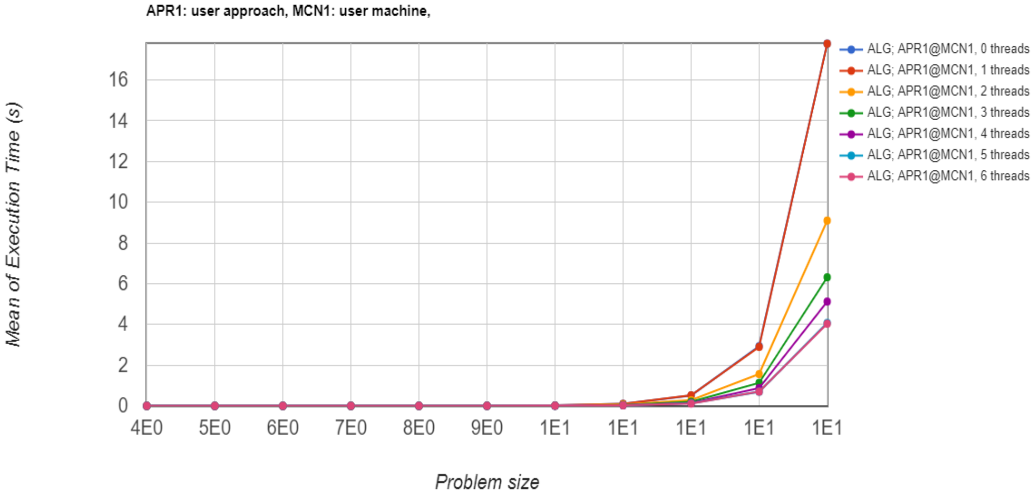
Since this algorithm is optimally designed to be present in middle ground of both the above methods, the execution time of this algorithm is half when compared with the above two algorithm's execution time. The speed up of this method increases with increases in number of execution time. The efficiency of the algorithm goes on decreasing as the processors count increases.

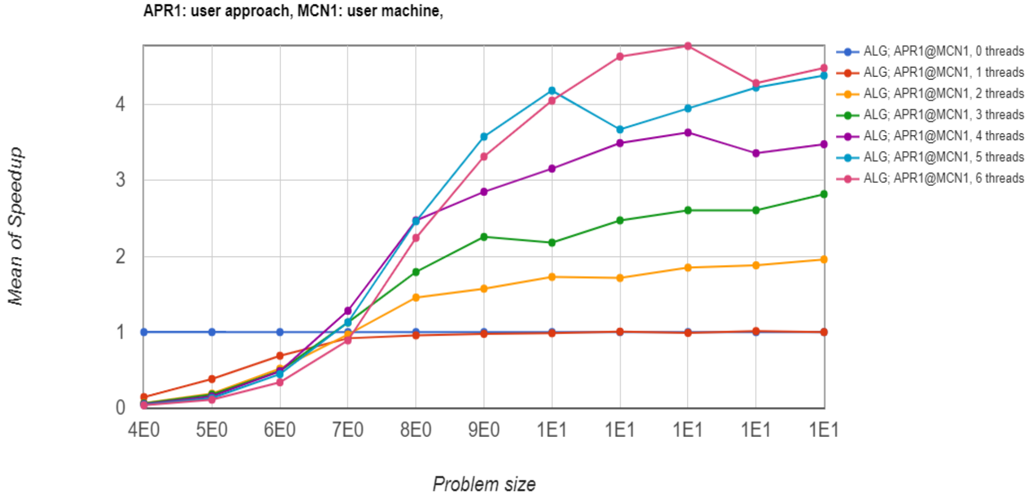




6.4 Method 4 : Parallelization of Backtracking + Bitmasking Algorithm

In this method we had tried to decrease the time of calculating function of whether the provided place is valid placement or not. We had used the backtracking + bitmasking technique to reduce total execution time whose information is provided in bit masking part of introduction section.





Results-quantitative Analysis-discussion

The Backtracking approach is a recursive approach which reduces the number of positions to be checked significantly. The time complexity in this case is found to be exponential estimated to be $O(2^n)$. The time complexity looks to be same as the previous method but the total time is actually affected because addition of bitmasking to the algorithm makes the algorithm faster for calculation of function which helps us to decide whether the given position is valid or not. But this utlimately increase the space complexity of the algorithm which is $O(n)+O(2n-1)+O(2n-1) = O(3n-2)$ here $O(n)$ is due to the column bitarray used to check the vertical attacks on the queen. Similarly $O(2n-1)$ is for each of the left and right diagonal for the checking the diagonal attacks on the queen.

The memory accesses done in this method are same as method-1 that is $2n^2$ accesses for each path in the grid, resulting in total of $2^n * 2n^2$ memory accesses. The space complexity and memory access is high in this case which results in increasing of execution time.

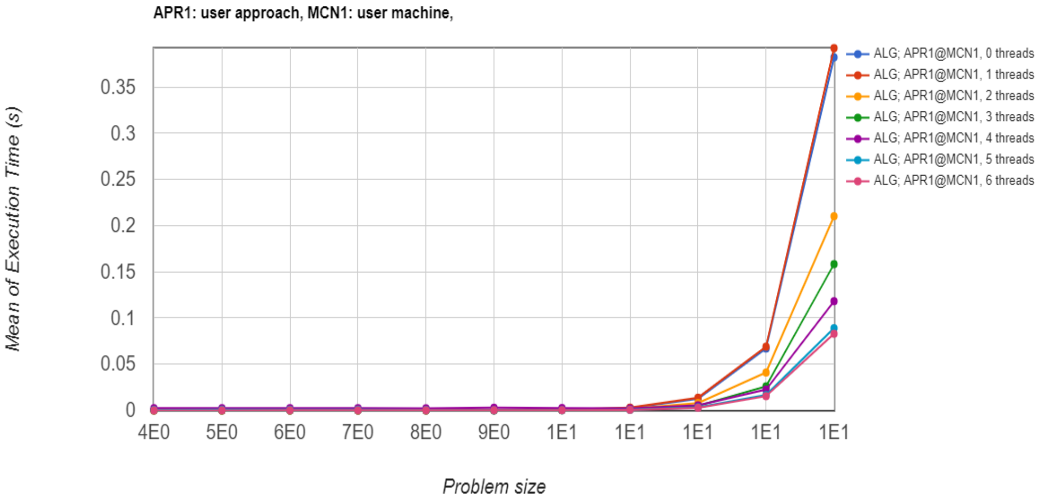
The execution time goes on decreasing as the number of processors are increased. As we have added bitmasking to the algorithm to make the algorithm faster for calculation of function which helps us to decide whether the given position is valid or not. But the graph contradicts to our assumptions because even though we have decreased the time complexity from $O(n)$ to $O(1)$ from the above function and space complexity increases since space is used to store three bit arrays which leads to the increase in cache misses and hence the total execution time increases. The speed up increases with increase in processors count similarly the efficiency decreases with increase in processors count.

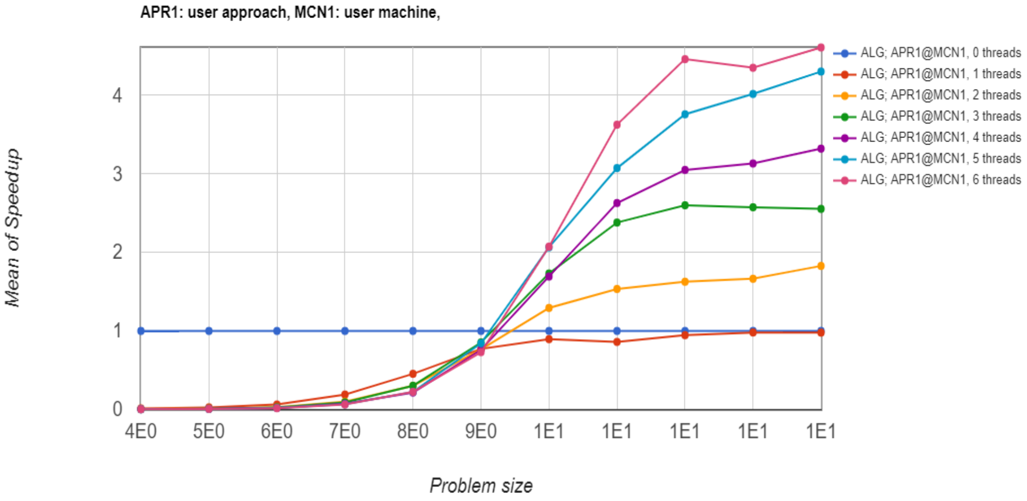
6.5 Method 5 : Parallelization of Backtracking + Bitmasking Algorithm

In this method we had also used backtracking + bitmasking but tried to even further optimize the method4 by using the integers in place of bit arrays which helps to reduce the space complexity this can ultimately helps us to decrease cache misses if there is an increase in the size of N.

Results-quantitative Analysis-discussion

The Backtracking approach is a recursive approach which reduces the number of positions to be checked significantly. The time complexity in this case is found to be exponential estimated to be $O(2^n)$. The time complexity looks to be same as the previous methods but the total time is actually different because addition of bitmasking to the algorithm makes the algorithm faster for calculation of function which helps us to decide whether the given position is valid or not. Moreover in this algorithm the large space complexity obtained in the method-4 is also optimized by using the an integer in the binary form in place of three bit array to check validity of the position which results in space complexity to be $O(1)$.





The memory accesses done in this method are same as method-1 that is $2n^2$ accesses for each path in the grid, resulting in total of $2^n * 2n^2$ memory accesses. Even though the memory accesses is very large but the memory-data that is accessed is always present in the cache memory which decreases cache misses and hence increases in the performance of algorithm. The total execution time of this algorithm is lowest because of two main reasons which are :-

1. Addition of bitmasking to the algorithm to make the algorithm faster for calculation of function which helps us to decide whether the given position is valid or not.
2. Using the integers to store the information of placements of queens in place of three bit arrays as used in model-4, helps us to save space in cache and also execute the validity condition faster.

The speed up of the model-5 is also good the increases with the increase in no of processors. Similarly, the efficiency of the algorithm decreases with the increase in number of processors due to overhead in process switching.

7 Conclusion

In conclusion we find by analyzing the speedup curves, that the method of parallelizing backtracking using bit masking in which we use integers instead of arrays is the most effective method among those we have used as it produces the most significant speedup and the lowest mean execution time. This algorithm also optimizes the space required.

8 Future Scope of Improvement

In the above methods for optimization purpose we have only considered different aspects like cache size and mirror-symmetry of N x N Board. In future we can add the rotational symmetries to the solution. Moreover, we can try to change the shared memory system to distributed memory system.

9 References

1. Wikipedia
2. Geeks for Geeks N queens Backtracking
3. Geeks for Geeks N queens Branch and Bound
4. N queens Backtracking + Bitmasking
5. Medium Article N queens Bit masking
6. For Parallel Programming of serial Codes