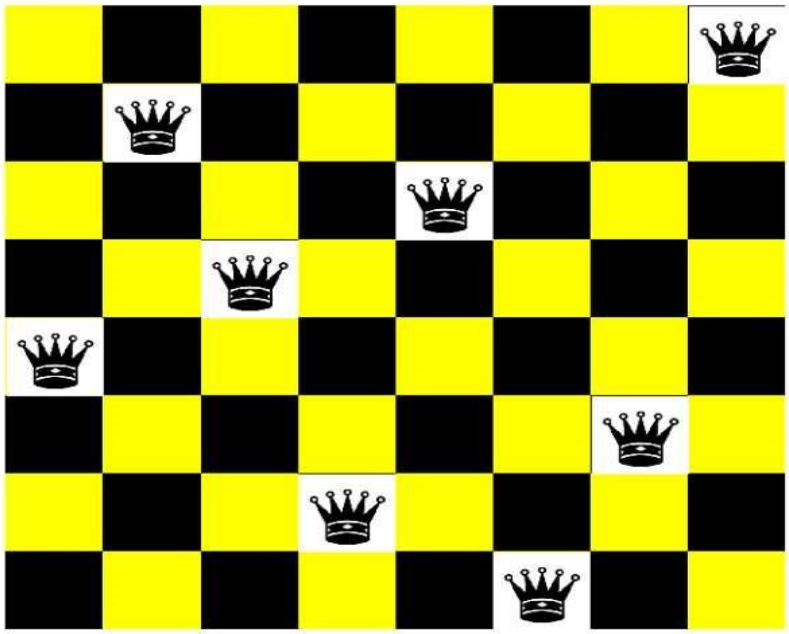


Parallel Genetic Algorithm for Optimizing 8-Queen Problem

ROLL NO

19L-2708 Zulqarnain

19L-2256 Waleed

19L-2745 Abdullah Saleem

Introduction

The eight-queen puzzle was initially proposed by the chess player Max Bazzel, in 1848. This puzzle is a problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other. The color of the queens is pointless in this puzzle, and any queen is assumed to be able to attack. This problem can be solved by multiple programming approaches, e.g., brute force, backtracking, or genetic algorithms. Now, there are a lot of serial solutions to this problem available, but they take a long time to execute in a worse case. What if we parallelized it using a genetic algorithm? If we can solve this problem using parallel genetic algorithms at a faster speed, then we can also map this problem to any other scenario. Hence, a solution requires that no two queens share the same row, column, or diagonal. This can also apply to different problems like timetable designing, puzzle-solving, cracking any password, etc.

Performance of Different Algorithms

1. BACKTRACKING ALGORITHM
2. BRUTE-FORCE ALGORITHM
3. SERIAL GENETIC ALGORITHM
4. COARSE-GRAINED PARALLEL GENETIC ALGORITHM

Now, we estimate the performance 8-queen problem by testing the different algorithms listed above.

BACKTRACKING ALGORITHM

Backtracking solves the problem by placing a queen on the first row of the board and then placing the second queen on the second row in a way it wouldn't conflict with the primary one. It continues putting the queens on the board row by using row till it places the last one on the n-th row. If it couldn't fill any tile on a row, it'd backtrack and change the placement of the previous row's queen.

Its time complexity is O(2^N).

Drawback

In the Backtracking algorithm, if we choose N=25, it will take 322.89 seconds to find the solution, and when N=26, it shall take forever!

Now, estimating the performance of the Brute-force algorithm.

BRUTE-FORCE ALGORITHM

Bruteforce algorithm solves the problem by generating all possible chess boards with 8 queens and later checks if finds its goal by any one of its moves. If not, it again generates the solution (which was not visited) and checks the solution again. It continues generating till it finds the answer.

Its time complexity is O(N^(2N)).

Drawback

In the Bruteforce algorithm, as the level of search increases, its time and complexity increase exponentially.

Its performance is even worse than Backtracking Algorithm.

Comparison of Obtained No. of Solutions and Execution Time between BT and BF

No. of Queens	BT No. of Solutions	BT Time (Sec)	BF No. of Solutions	BF Time (Sec)
1	1	0.000000	1	0.000000
2	0	0.000000	0	0.000000
3	0	0.000000	0	0.000000
4	2	0.000000	2	0.000000
5	10	0.000000	10	0.000000
6	4	0.000000	4	0.000000
7	40	0.054945	40	0.054945
8	92	0.109890	92	0.164835
9	352	0.384615	352	0.714286
10	724	1.153846	724	1.978022
11	2680	5.164835	2680	9.450549
12	14200	32.252747	14200	58.736264

SERIAL GENETIC ALGORITHM

The need of the hour is to have such an algorithm that will perform calculations much faster with maximum performance. And Genetic Algorithm is the key. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. The major steps involved:

- Population
- Crossover
- Mutation
- Selection

As a serial genetic algorithm uses the random searching technique, its time complexity varies every time. But this searching technique proves way too faster than the above-listed algorithms.

Drawback

A genetic algorithm doesn't guarantee the answer. In some cases, it stuck at some fitness value. Hence, it will take a long time to back up it.

COARSE-GRAINED PARALLEL GENETIC ALGORITHM

As genetic algorithms do not depend on each other, as a result, they can run in parallel, so we can run multiple threads for which we maximize the performance by assigning different threads to run mutation, crossover, selection, and population randomization as they are independent tasks.

In most cases, its time complexity becomes way better than a serial genetic algorithm.

Drawback

To make a genetic algorithm parallel requires carefully managing all the parallel sections with synchronization. (if require)

The following analysis shows the performance of serial vs parallel genetic algorithms.

QUEENS	CPGA/S	SGA/S	ACCELERATE RATIO
8	0.001000	0.001000	1.000
16	0.001000	0.036000	3.600
32	0.017000	0.031000	1.832
64	0.111000	0.130000	1.182
128	0.569000	1.662000	2.921
256	4.054000	18.853000	4.650
512	39.494000	87.691000	2.220

Result So Far

With the help of the above analysis, we notice a significant increase in performance for solving 8-queen problems if we go for the genetic algorithm technique rather than Brute-Force or Backtracking Algorithms.

Moreover, if we use a coarse-grained parallel genetic algorithm, the performance even escalates.

Example

In the CPGA model with 4 processors, we can get a **2.485 average to accelerate ratio**, the accelerate ratio is rising tendency with the increase in Queen Numbers

We further estimate the performance by testing different omp-parallel technique like.

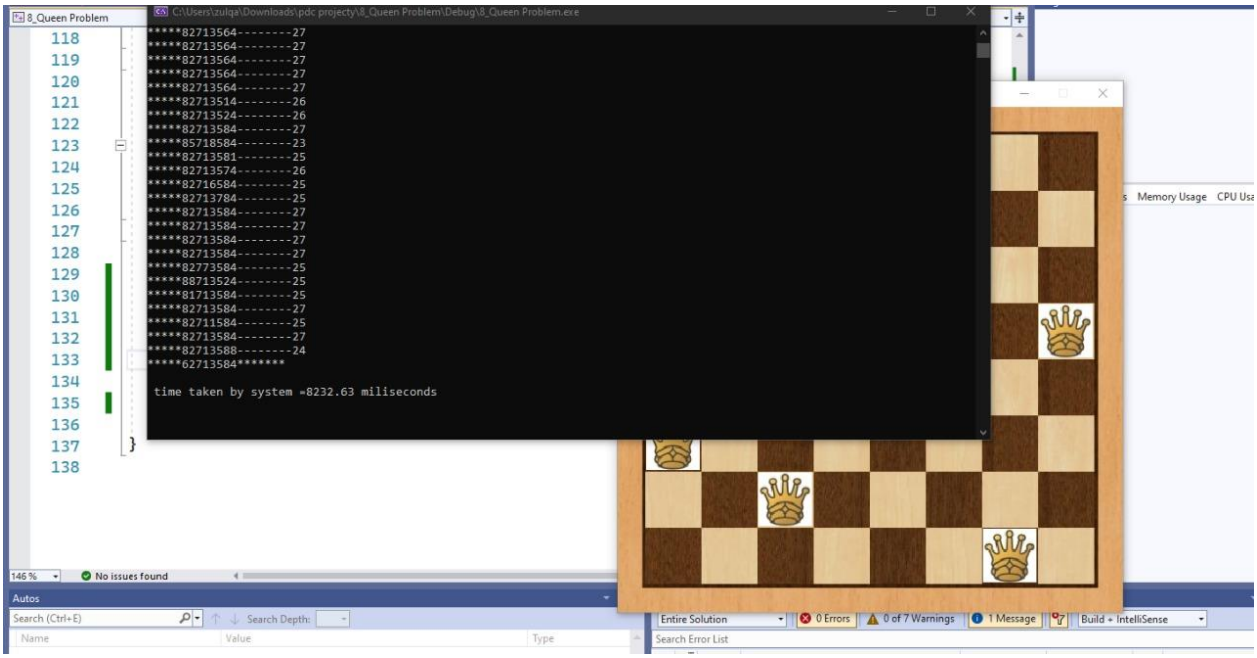
a) STATIC
b) DYNAMIC
c) GUIDED
d) RUNTIME

~ (STATIC) ~

#pragma omp parallel for schedule (static, chunk-size)

By default, OpenMP statically assigns loop iterations to threads. When the parallel for the block is entered, it assigns each thread the set of loop iterations it is to execute.

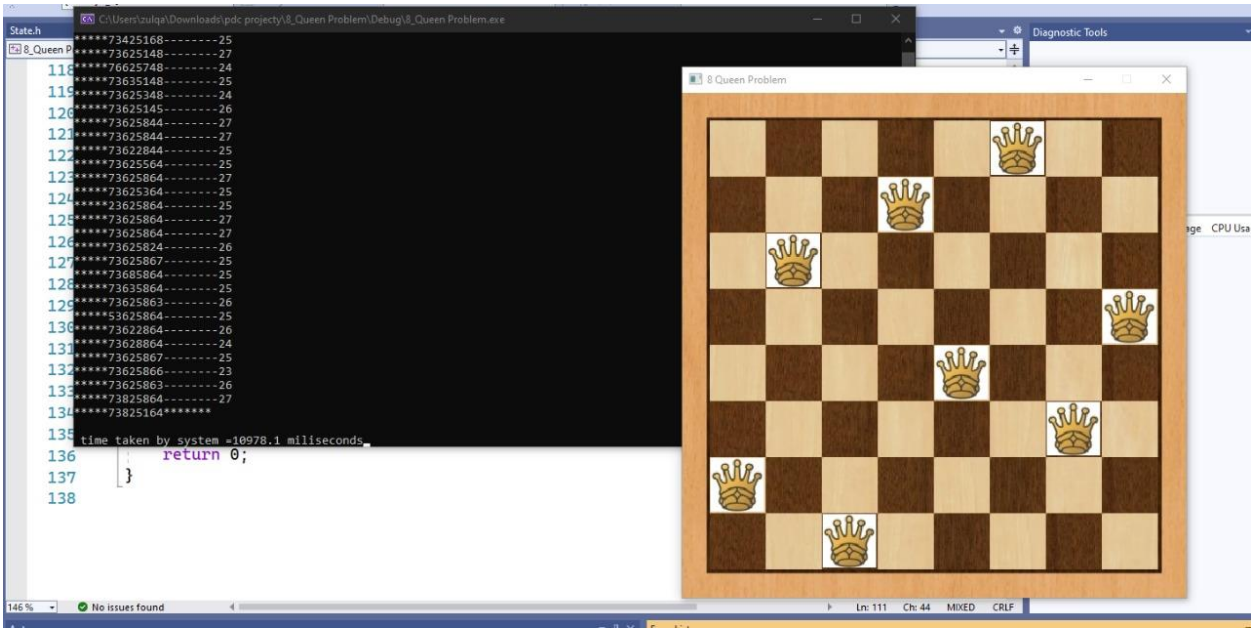
If you do not specify a chunk-size variable, OpenMP will divide iterations into chunks that are approximately equal in size, and it distributes chunks to threads in order (that is why the static method is different from others.)



~ (DYNAMIC) ~

#pragma omp parallel for schedule (dynamic, chunk-size)

After each iteration, the threads must stop and receive a new value of the loop variable to use for its next iteration

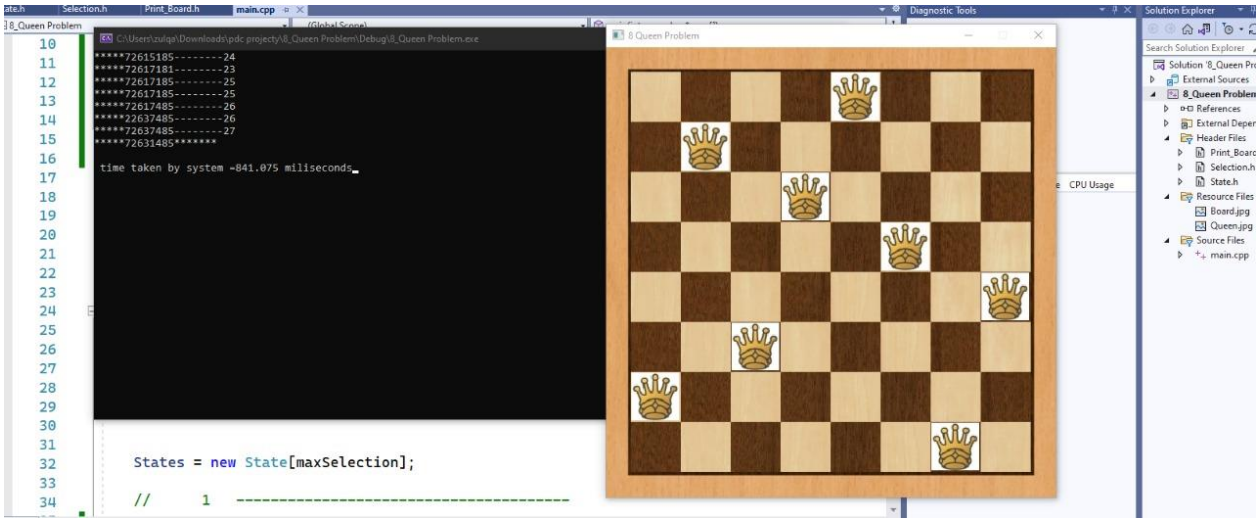


~ (GUIDED) ~

#pragma omp parallel for schedule (guided, chunk-size)

Guided is similar to a dynamic schedule, except that the chunk size changes as the program runs. It begins with big chunks but then adjusts to smaller chunk sizes if the workload is imbalanced.

The size of a chunk is proportional to the number of unassigned iterations divided by the number of the threads, and the size will be decreased to chunk-size (but the last chunk could be smaller than chunk-size).



~ (RUNTIME) ~

Depending on the environment variable OMP_SCHEDULE we set in the command line

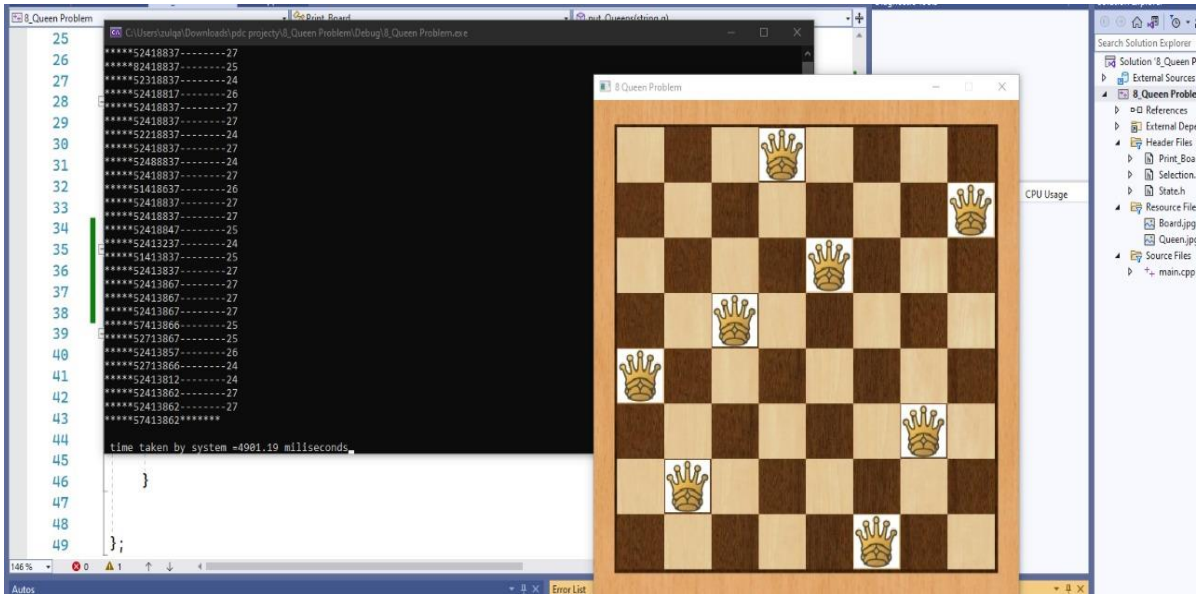
OpenMP for automatically splits for loop iterations for us.

But, depending on our program, the default behavior may not be ideal.

For loops where each iteration takes roughly equal time, static schedules work best, as they have little overhead.

For loops where each iteration can take very different amounts of time, dynamic schedules, work best as the work will be split more evenly across threads.

Increasing the chunk size makes the scheduling more static, and decreasing it makes it more dynamic.
Specifying chunks, or using a guided schedule provides a trade-off between the two



The performance analysis of the above omp-commands areas.

N= number of iterations of the parallel loop

P= number of threads

C=user-specified chunk size

Name	Type	Chunk	Chunk Size	# of Chunks	Static or Dynamic	Computer Overhead
Simple Static	simple	no	N/P	P	static	lowest
Interleaved	simple	yes	C	N/C	static	low
Simple Dynamic	dynamic	optional	C	N/C	dynamic	medium
Guided	guided	optional	decreasing from N/P	fewer than N/C	dynamic	high
Runtime	runtime	no	varies	varies	varies	varies

Conclusion

In the end, we notice a significant increase in performance for solving 8-queen problems. We will are able to analyse the outcomes of different mapping techniques on performance. And we observe how open-mp libraries help us to use a genetic algorithm to split the components and run in parallel to boost performance.

References

[1] "Eight queens puzzle - Wikipedia", En.wikipedia.org, 2022. [Online]. Available: <https://en.wikipedia.org/wiki/Eight_queens_puzzle> [Accessed: 15- Apr- 2022]

[2] "Genetic algorithm vs. Backtracking: N-Queen Problem", Medium, 2019. [Online]. Available: https://towardsdatascience.com/genetic-algorithm-vs-backtracking-n-queen-problemcdf38e15d73f [Accessed: 15- Apr- 2022]