Serial and Parallel execution of Genetic N Queens Algorithm

Riddhi Singhal
Dept. of Computer Science and Engineering,
Ramaiah Institute of Technology
Bangalore, India

K Divyasri
Dept. of Computer Science and Engineering,
Ramaiah Institute of Technology
Bangalore, India

Mallegowda M
Dept. of Computer Science and Engineering,
Ramaiah Institute of Technology
Bangalore, India

***Abstract* - Serial processing is the one in which one task is finished at a moment and all the tasks are carried out by a single processor sequentially. The processor receives lists of tasks, and completes each one one at a time while delaying the completion of all others. Parallel processing is the one in which multiple tasks are carried out simultaneously by different processors.There are multiple processors at play. Multiple task queues exist in the operating system, and various processors work on several tasks at once. Large-scale computations can be completed more quickly due to parallel computing, which makes use of multiple processors at once. Programmes can now be executed in much less wall-clock time because of the time savings offered by parallel computing. The N Queen problem is the placement of N chess queens on a N–N chessboard without any queens attacking one another. No two queens can therefore be found in the same row, column, or diagonal to find a solution. This is a comparison study of the N Queen solution based on genetic algorithms in serial and parallel execution environments.**

Keywords - Serial and Parallel programming, N Queens problem, Genetic algorithm.

I. INTRODUCTION

When processing in serial mode[1], each task is finished by the CPU individually. Following that, the other tasks are completed in a particular order. An operating system's programmes have a number of tasks that they must accomplish. One processor is used for serial processing. The processor must carry out each of these responsibilities, but they are all completed individually. Other tasks are held in a queue until the processor completes the one it is working on. So, each task is completed one at a time. Data transfers happen piecemeal because the CPU is under increased strain. The term "sequential processing" or "serial processing" is thus used to describe this process.

Multiple processors are employed in parallel processing[2]. Each processor's assigned tasks are finished concurrently. Data is sent in bytes as a result, and each CPU has less work to do. The bus is used by the processors for primary memory access and interprocessor communication. Every CPU operates with its own local data. Since each processor functions independently, the failure of one does not affect the operation of the others. As a result, parallel processing increases throughput while simultaneously increasing reliability.

A genetic algorithm[3] is a search heuristic that is based on natural selection. It is used to solve optimal or nearly optimal solutions to constrained and unconstrained optimization problems which otherwise would take an eternity to solve. It uses an iterative approach to find the optimum result, selecting the best answer from a range of best answers.

NP-complete problems are a set of problems that can be solved by a non-deterministic turing machine in polynomial time. In other words, these are problems for which no effective solution has been discovered. They have a high level of complexity of the order of 2n or n!, which makes it impossible to solve them using deterministic methods in real time.

One of the simplest NP-Complete problems might be the N-Queens[4] Completion problem. It entails arranging n queens on a n × n chessboard so that no two queens square off against one another, that is, there is no intersection between them along any of the three axes-horizontally, vertically or diagonally. Because we are choosing whether or not to place a Queen there, the time complexity is O(n2). The N-queens puzzle has an O(NXN) time complexity for the worst-case "brute force" solution. This means that it will search for N queens in N positions throughout a NxN board, N times.

II. RELATED WORK

In [5] Jianli et al. concentrated on how to speed up the evaluation function to resolve variation 2 of the N Queens problem using heuristic techniques. To evaluate potential answers for the N Queens problem, three parallel CPU strategies, four parallel GPU schemes, and a serial scheme are offered. The experimental results show that the coarse-grained GPU technique outperformed its single-threaded CPU counterpart by a maximum of 307 times for a large-scale N-Queens problem. The speedup is increased to 9.3 when the coarse-grained GPU technique is used to solve the N-Queens problem variant 2 using simulated annealing with a problem size of no more than 3000. If, unlike the existing one, device memory is reused in numerous evaluation function calls, the performance of the coarse-grained GPU technique can be significantly enhanced. By using NVidia NVVP to read the hardware counters in the GPU, assess the program's micro performance bottleneck, and optimise some CUDA kernel configuration parameters, it is possible to further boost the efficiency of GPU methods. In order to lessen the cost of thread management and boost performance for the dynamic parallel scheme, some sub kernel launch random events can be cancelled using bypass technology. By using NVidia NVVP to monitor the hardware counters in the GPU, assess the program's micro performance bottleneck, and optimise some CUDA kernel configuration parameters, it is possible to further boost the efficiency of GPU methods. In order to lessen the cost of thread management and boost performance for the dynamic parallel scheme, some sub kernel launch random events can be cancelled using bypass technology. The performance of the algorithm can then be further enhanced by combining thread-level parallel technologies with instruction-level parallelism techniques.

In [6], Thouti et al. used the OpenCL programming model to explore the cost of solving the N-Queens issue on GPGPU architecture. It was discovered that using local memory instead of global memory for memory pooling significantly speeds up OpenCL. When explicit global atomics were used instead of the OpenCL default, the N-Queen problem performed better. When global atomics was enabled with local memory, the performance was significantly improved. Future work includes using these techniques to tackle other issues like sparse matrix-vector multiplication, the Euler problem, shortest paths, and other sorting methods.

Umbarkar et al. [7] summarised the research on the application of the Parallel Genetic Algorithm (PGA) on various parallel computing architectures. A brief summary of theoretical developments and computing trends was provided, focusing on population diversity in PGA. With certain of the applications for Massive Parallel Processing (MPP), Grid, and cluster-fine, medium, and coarse-grained Genetic Algorithms, superliner performance was demonstrated. PGA lowered function evolution, improved solution quality, explored massive population sizes, and successfully addressed challenges with big ranges and large dimensions when implemented on parallel systems. It has been noted that while GAs are utilised to address cloud optimization issues, they are not actually deployed on the cloud.

In order to solve the N-queens issue on a multicomputer platform, Lazarova et al. [8] examines the effectiveness of parallel genetic algorithms. Utilising a ring topology and an island-based parallel genetic method, randomly picked chromosomes move in both directions dynamically. It is possible to implement the algorithm utilising flat (pure MPI) or hybrid (MPI+OpenMP) programming models. According to both the parallel workload (board size) and cluster size, the findings demonstrate the parallel computational model's strong scalability. The application scales up and down with the size of the multicomputer with roughly proportional speedups and great parallel system efficiency. Performance

analysis demonstrates that the hybrid parallel programming paradigm more effectively utilises the parallel hardware resources of the target multicomputer platform. Future work should investigate methods for regulating certain genetic algorithm parameters, such as migration size and mutation rate, on the performance of the parallel computational model.

Turky et al. [9] adopted a genetic algorithm to solve the N-Queens problem. The tests for the N-Queens problem were carried out on a range of populations and board sizes. Each result is presented as the average of ten runs with the same board size and population.On average, it took lO23.845 seconds to find a solution for the biggest board size, n = 2000. The technique was written in Java, and simulations were ran on an AMD Athlon computer running Windows XP 2002 at 1.92 GHz with 512 MB of RAM. According to tests, a genetic algorithm with a repair function can find several answers for a given number of queens at various times. The best time to locate an ideal solution based on the repair function suggests this.

III. PROPOSED WORK

Genetic Algorithms[11] are a class of search and optimization tools motivated by evolution. The most common evolutionary processes used by evolutionary algorithms are selection, variation, and replacement operations.

Initialization, which generates a starting population of potential solutions, kicks off the evolutionary process. The most common initialization technique is to simply generate a population of binary strings that have been initialised at random. A selection procedure selects a group of parents from the parent population at each chronological step based on each individual's fitness values so that the fittest ones will have a higher likelihood of carrying on genetic information to succeeding generations. After the selection procedure has fully filled the parent population, a child population is produced that will serve as the foundation for the upcoming generation. Variation operators are used to create this child population from participants of the parent population. Crossover and mutation are the two most popular of these techniques.

Crossover function: The cross over function plays a crucial role in producing superior offspring for attaining the global optimum for a non-convex function. Parents are separated into pairs by crossover. Then, portion of the genetic material from each of these parents is exchanged.

Mutation: The common approach in evolutionary algorithms for adding new genetic material to the population is mutation. To prevent the process from being trapped at a local optimum, mutation is crucial in genetic algorithms.

IV. METHODOLOGY

The following steps were followed for implementing genetic N Queens algorithm-

1. As part of population initialization, we first initialise a randomly generated population of 500 chromosomes. Each chromosome is actually an unique permutation of values in a vector of size N. (1, 2, 3... N)
2. After creating a random population of potential solutions, the population's chromosomes are sorted and ranked according to fitness values. The fitness scores of the individuals are also determined.
3. Every chromosome's fitness has been designed so that k queens on the same diagonal position will increase the fitness value by k-1 points. All of these data points are added up to determine the fitness value.
4. A selection operator will determine how many chromosomes will be passed on to the next generation, favouring the better individuals depending on their standing in the population.
5. To produce offspring whose fitness values will be calculated simultaneously, selected chromosomes will participate in crossover operation as parents. Due to the perception that higher crossover probabilities produce offspring with higher fitness values, they are typically maintained at a high level.

6. Next, a mutation operator that randomly alters a few chromosomes is applied to new people depending on a mutation probability. The probability of mutation is typically maintained low.
7. Children who have been evaluated, along with their parents, make up the population of the following generation.

Once the difference between the two optimal solution from subsequent generations is below a specified tolerance limit, steps 2 to 5 are iterated until a predetermined amount of generations, or iterations, have been reviewed or the solution improvement rate meets a predetermined threshold. During this process, the population initialization, fitness function, crossover, and mutation operators were used.

Fitness Function: Consider the N x N configuration of the chess board (c1, c2, c3, etc.). First, we'll determine if many Queens are located on the same (↗) directed diagonals.This happens if,
For p and q $\in$ {1,2 … N}
(ci − p) == (cj − q)

The subsequent check, similar to the previous check, the following check will determine if many Queens are on the same (↖) directed diagonals, this happens if
For p and q $\in$ {1,2 … N}
(ci + p) == (cj + q)

In the event that any of the aforementioned circumstances arise, we tally the fitness value. Which demonstrates unequivocally that the fitness value of the fittest chromosome is zero (0).

V. EXPERIMENTS AND RESULTS

Fig. 1 depicts the plot between number of threads and execution time (in seconds) for both serial and parallel execution for the N Queens problem[11] with N being set to 8. As the number of threads increases, the time required for series execution increases and for parallel execution decreases. This is due to the fact that when a thread is executed in a series, each following thread must wait for the completion of its preceding thread's execution before continuing. Due to this, the process is slow and time-consuming. In contrast, each thread in the process can operate simultaneously on a different processor during parallel execution, which eliminates the wait time for threads. Thus, simultaneous execution takes less time to complete.

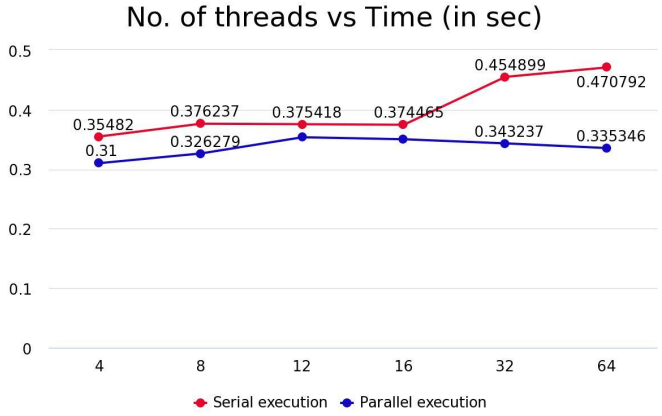

Fig. 1 Plot between the number of threads and execution time (in seconds)

TABLE 1. Serial and Parallel execution time (in seconds)

Number of threads	Time for Serial execution (in sec)	Time for Parallel execution (in sec)
4	0.35482	0.31
8	0.376237	0.326279
12	0.366418	0.35351
16	0.374465	0.350131
32	0.454899	0.343237
64	0.470792	0.335346

VI. CONCLUSION AND FUTURE WORK

In many cases, the size and/or complexity of the problem makes it nearly impossible to solve it on a single computer, specifically with the limited memory that is available. This issue can be resolved using parallel computing, which also quickly and efficiently addresses larger issues. The classic N-queen puzzle is one of computer science's most

difficult problems. Backtracking, which has a very high time complexity, is a well-known approach to solve the N Queen puzzle. As n rises, complexity increases exponentially. Metaheuristics like the Genetic Algorithm can be used to solve the N-Queens problem in a sensible timeframe.

Future research should look into the N Queens problem execution time for greater values of N while concurrently increasing the number of threads.

REFERENCES

[1] Verwey, W.B., Shea, C.H. & Wright, D.L. A cognitive framework for explaining serial processing and sequence execution strategies. *Psychon Bull Rev* 22, 54–77 (2015). https://doi.org/10.3758/s13423-014-0773-4

[2] Kasahara, H., Obata, M., Ishizaka, K. (2001). Automatic Coarse Grain Task Parallel Processing on SMP Using OpenMP. In: , *et al.* Languages and Compilers for Parallel Computing. LCPC 2000. Lecture Notes in Computer Science, vol 2017. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45574-4_13

[3] Dozier, G.; Bowen, J.; Bahler, D. (1994). *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence - Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm. , (), 306–311.* doi:10.1109/icec.1994.349934

[4] I. Martinjak and M. Golub, "Comparison of Heuristic Algorithms for the N-Queen Problem," *2007 29th International Conference on Information Technology Interfaces*, 2007, pp. 759-764, doi: 10.1109/ITI.2007.4283867.

[5] Jianli Cao, Zhikui Chen, Yuxin Wang, He Guo, "Parallel Implementations of Candidate Solution Evaluation Algorithm for N-Queens Problem", *Complexity*, vol. 2021, Article ID 6694944, 15 pages, 2021. https://doi.org/10.1155/2021/6694944

[6] Thouti, K., & Sathe, S. R. (2012). Solving N-Queens problem on GPU architecture using OpenCL with special reference to synchronisation issues. 2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing. doi:10.1109/pdgc.2012.6449926

[7] A. J. Umbarkar, M. S. Joshi. (2013). Review of Parallel Genetic Algorithm based on Computing paradigm and diversity in search space. ICTACT JOURNAL ON SOFT COMPUTING, JULY 2013, VOLUME: 03, ISSUE: 04. ISSN: 2229-6956

[8] Lazarova, Milena. (2008) “Efficiency of parallel genetic algorithm for solving N-queens problem on multicomputer platform” 9th WSEAS International Conference on EVOLUTIONARY COMPUTING (EC’08), Sofia, Bulgaria, May 2-4, 2008 , 51-56 pages

[9] A. M. Turky and M. S. Ahmad, "Using genetic algorithm for solving N-Queens problem," *2010 International Symposium on Information Technology*, 2010, pp. 745-747, doi: 10.1109/ITSIM.2010.5561604.

[10] Crawford, Kelly D. (1992). [ACM Press the 1992 ACM/SIGAPP symposium - Kansas City, Missouri, United States (1992..-..)] Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing technological challenges of the 1990's - SAC '92 - Solving the n-queens problem using genetic algorithms. , (), 1039–1047. doi:10.1145/130069.130128

[11] M. Plauth, F. Feinbube, F. Schlegel and A. Polze, "Using Dynamic Parallelism for Fine-Grained, Irregular Workloads: A Case Study of the N-Queens Problem," 2015 Third International Symposium on Computing and Networking (CANDAR), 2015, pp. 404-407, doi: 10.1109/CANDAR.2015.26.