Parallel Execution and Optimizations of a Backtracking Algorithm for a Connect Flow Puzzle Problem (Backtracking Algorithm)

MingYao Lin
mylsonge@gmail.com
NYCU, parallel programming course
HsinChu, Taiwan

YiJyun He
moonrain716s@gmail.com
NYCU, parallel programming course
HsinChu, Taiwan

ABSTRACT

This study focus on optimizing the solving process of connect flow problems by parallel execution by recursive backtracking algorithm. The rule of this problem is being able to connect all pairs of dots with colour corresponded on a 2D grid. Meanwhile, none of the lines which are join together can cross over each other. We compare and analyze the different performance and effort of two types of Shared-Memory model, POSIX Threads(Pthreads) and OpenMP. The performance is scored by the time for solving Connect Flow Puzzle Problems through Backtracking Algorithm under two different models and structure.

1 INTRODUCTION/MOTIVATION

Today most computers have multi-core processors to enhance their performance. As long as keeping up efficiency by parallel executions, Software can take advantages of potential processing capability of multi-core system.

This study focus on the different performance and effort of two types of typical Shared-Memory model, POSIX Threads(Pthreads) and OpenMP. The rule of this problem is being able to connect all pairs of dots with colour corresponded on a 2D grid (as figure 1). Meanwhile, none of the lines which are join together can cross over each other.

Backtracking Algorithm is a classic problem-solving technique for Eight Queens Problem and will be applied to deal with the puzzle problem in this project. It will explore one path at a time, going deeper and deeper into the solution space until it finds a specific solution or exhausts all possibilities.

If we process Backtracking Algorithm in sequential program, it goes without saying that it will consume a large amount of time, especially for complex problems. In this study, we will try to apply the Backtracking Algorithm through parallel execution and study if it can solve a classic Connect Flow Problem with better performance.

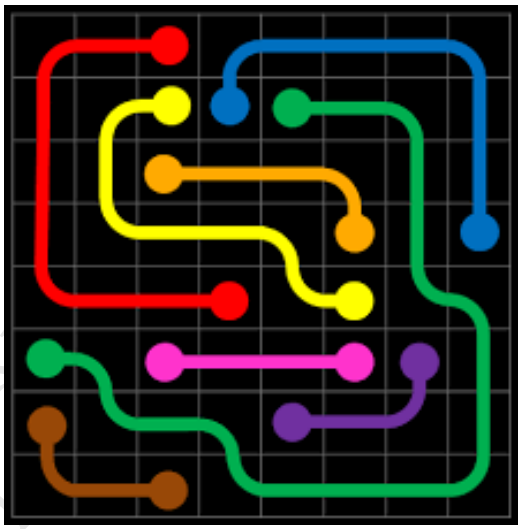

Figure 1: Connect Flow Puzzle Problem

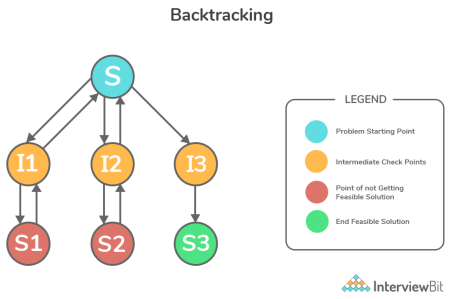

Figure 2: Backtracking Alogrithm

2 PROPOSED SOLUTION

The Connect Flow Puzzle Problem in this project is designed as a 2D grid with N pairs of dots with color corresponded (N colors, N*N table). To parallel Backtracking Algorithm for solving the Puzzle Problem, We will design two different parallel frameworks.

The first framework is to use a work-stealing algorithm, in which each thread starts from a different direction of one of the starting point and attempts to pair all colors and find a solution to cover

2023 Parallel Programming Course, NYCU, Hsinchu, Taiwan.

ACM ISBN ... $15.00
https://doi.org/

the entire grid board. This enables the searching process to explore multiple paths at the same time.

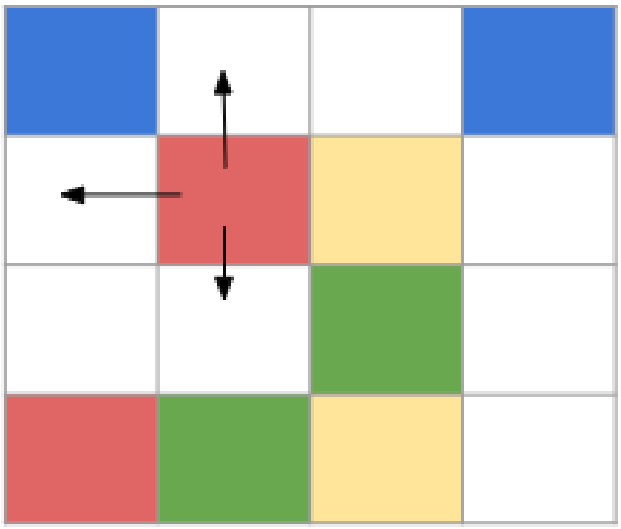

Figure 3: The first framework

The second one is to parallel each searching process of possible path of one color from its starting point to its end point. It limits the bottleneck of waiting for the other pair of dots being solved. After each thread finished their backtracking job, we collect the solved set from each color. Then we checked the combination which is the answer to the connect flow problem. It also needs the backtracking technique to choose one path from the possible solution of each color. The set of the possible solution is expected to be less amount than the n-step deep searching with different directions from the starting point since we reduce the branching factor – the average number of choice per step.

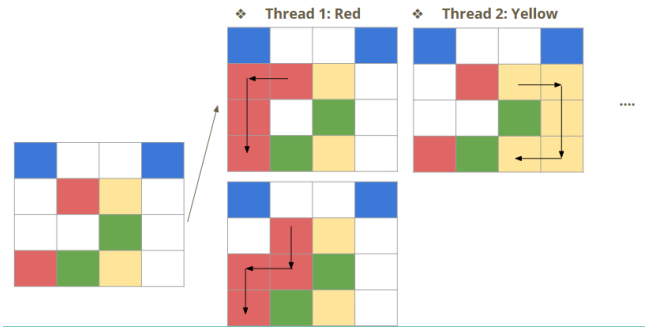

Figure 4: The second framework

On the other hand, threading pool would be used as a managed collection of threads that are available to perform tasks.

For each framework, we will execute them with two type of Share-Memory Model, Pthreads and OpenMP. Pthreads is a very low-level API for working with threads. OpenMP is much higher level and much more easily scaled than pthreads. That means there are four conditions for solving the puzzle problem.

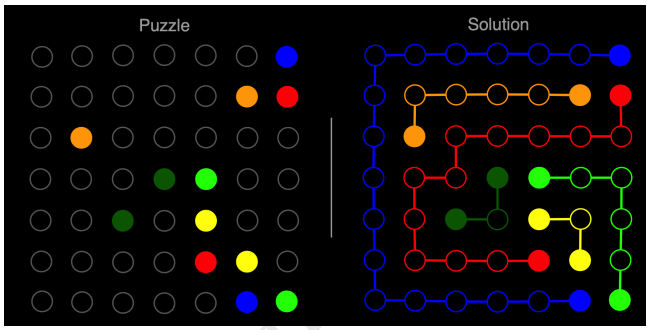

Figure 5: Solution to a Connect Flow Puzzle Problem

3 EXPERIENTIAL METHODOLOGY

The program run on Ubuntu 20.04, Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz with 4 cores and 15GB of RAM.

The two framework support randomized starting point and ending point of each pair of dots, so it can generate different solving or searching path for each test iteration. We measure our run time on N = 1 . . . 4 threads, and run about 10 iterations to obtain an average results of each inputs. We evaluate performance by comparing run time for returning the a solution from each condition.

The input puzzle from N = 3 to N = 8, as seen on Figure 6 to 9. The value "1" represents the boundary of the puzzle, and each value greater than one represents one dot of a color. We ran the program with two frameworks and two parallel languages.

```
1 1 1 1 1
1 2 4 4 1
1 0 0 3 1
1 2 0 3 1
1 1 1 1 1
```

Figure 6: Input Data - 3x3

```
1 1 1 1 1 1
1 4 0 0 4 1
1 0 2 3 0 1
1 0 0 5 0 1
1 2 5 3 0 1
1 1 1 1 1 1
```

Figure 7: Input Data - 4x4

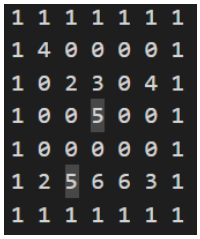

Figure 8: Input Data - 5x5

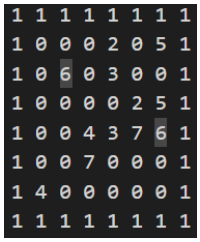

Figure 9: Input Data - 6x6

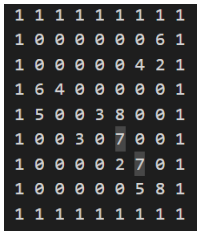

Figure 10: Input Data - 7x7

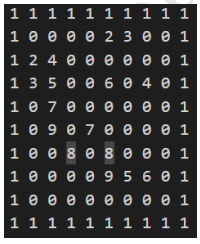

Figure 11: Input Data - 8x8

The input puzzle of N = 9 and N = 10, as seen on Figure 10 and 11. We tested the input with N = 9 and N = 10 only by sequential framework and the first framework to choose one direction of starting point for each thread. Because the second framework will store the collection set of possible path as a binary number which needs N * N bits for Input data N*N. Therefore, the second structure cannot be executed when N > 9, since 9 * 9 bits and 10 * 10 bits are out of the range of unsigned long long integer.

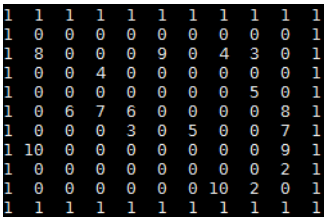

Figure 12: Input Data - 9x9

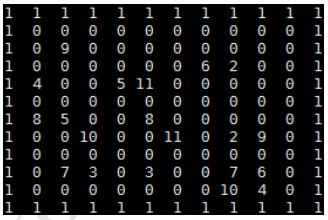

Figure 13: Input Data - 10x10

4 EXPERIENTIAL RESULT

4.1 Sequential v.s. Parallel

The Figure 14 shows that the sequential program always get the answer faster than parallel program when N is from 3 to 8, regardless of the framework or language. It might because that the parallel solution costs more time to divide the question into smaller and independent sub-question and assign each sub-question to a separate processor with their own memory space.

We test the input with N = 9, 10 and compare the time spent by sequential version and by the first framework. The first framework running with pthread performed better when N increased. The second framework still took more time than sequential version did but the gap between them became smaller with N increased.

4.2 Pthread v.s. OpenMP

The Figure 14 shows that the openmp version always ran faster than pthread version did when N is from 3 to 7. The reason might be the script of thread pool of pthread is designed by ourselves and it costs more time to allocate memory and pass parameter.

We tested the input with N = 9, 10 and compared the run time of pthread version and openmp version. The pthread ran faster and the performance improved.

5 RELATED WORK

5.1 Solve Eight Queen Problem with C-MPI

The 8-queens problem is a classical combinatorial problem in which it is required to place eight queens on an 8 x 8 chessboard so no two can attack. That means no two of them are in the same row, column, or diagonal. A trivial generalization of this problem requires one to place n queens on an n x n chessboard so that now two can attack. In this ongoing work, researchers described a parallel algorithm for generating solutions of the n-queens problem and its implementation in C-MPI.

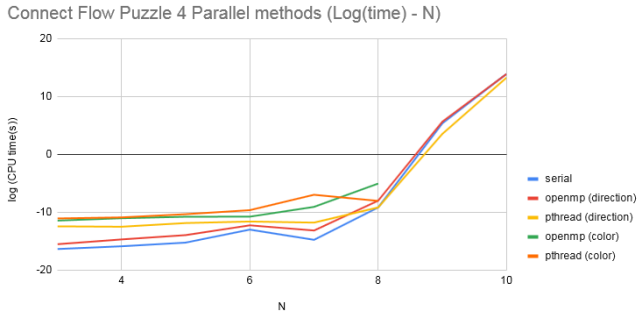

Figure 14: Connect Flow Puzzle 4 Parallel methods (Log(time) - N)

5.2 Solve N-Queen problems by paralleling backtrack search algorithm with .NET Framework

There was a project about parallel version of the generic backtrack search algorithm. It is used to find all or just a single solution to the N-Queen problem. Furthermore, various new heuristics and optimizations are explored. The project also explored how the .NET Framework may aid in the design on parallel programs

5.3 Pthread vs OpenMP

Researchers once paralleled three algorithms, Matrix multiplication, Quick Sort by using OpenMP and Pthreads. The results shows that OpenMP does perform better than Pthreads in Matrix Multiplication and Mandelbrot set calculation but not on Quick Sort. It was because OpenMP has problem with recursion and Pthreads does not. OpenMP wins the effort required on all the tests but because there is a large performance difference between OpenMP and Pthreads on Quick Sort, OpenMP cannot be recommended for paralleling Quick Sort or other recursive programs.

6 CONCLUSION

There are four conditions in this study based on two parallel languages and two frameworks of program. Comparing the sequential program and parallel one, time consumption of sequential program is always less than parallel program's when puzzle size is less than 8*8. It's different from our previous assumption that parallel process can have less run time for finding an answer to connect flow question by exploring multiple paths at the same time without tracking back and re-searching new potential solution. It might because the parallel solution costs more time to divide the question into smaller and independent sub-question and to assign each sub-question to a separate processor with their memory space, but when inputs become larger, the time we save from exploring multiple paths at the same time can cover the time we spent on task division and memory assignment.

On the other hand, the program structure we designed limit the amount of thread on equal or less than 4 might limit the best performance of the parallel execution, and the second structure even limit the input size and executable range of problem. Therefore, our future research will focus on algorithm optimization. We expect to have obviously better performance by overcoming the current constraints of input size and thread amount.

As for two frameworks of program, OpenMP is considered sufficient and user-friendly when paralleling program, however we need to consider the time for compiler to deal with for command(#pragma); on the other side, Pthread is a very low-level API library for working with threads, so it might improve the performance more when testing with large scale because programmer can control more details of implement than Openmp does.

REFERENCES

Lorna E. Salaman Jorge.(2001)."A Parallel Algorithm for the n-Queens Problem",Department of Mathematics, University of Puerto Rico, Mayaguez Campus, Mayaguez, Puerto Rico

Henrik Swahn.(2016)."Pthreads and OpenMP: A performance and productivity study", Faculty of Computing, Blekinge Institute of Technology, SE-371 79 Karlskrona Sweden