ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

An ISO 3297: 2007 Certified Organization Volume 5, Special Issue 11, May 2016

International Conference on Innovations in Engineering and Technology (ICIET'16)

6th & 7th April 2016

Organized by

K.L.N. College of Engineering and Technology, Madurai, Tamil Nadu, India

Parallelization of Maze Generation and Solving in Multicore Using Openmp

R.Muthuselvi* ,Sindhuja.A**,P.K.D.Sridevi***

Dept. of Computer Science Engineering, Kamaraj College of Engineering and Technology, India

ABSTRACT: Multi-core processing is a growing industry trend as single core processors rapidly reach the physical limits of possible complexity and speed. Most current systems are multi-core. In order to use each and every one efficiently there is a need to move towards parallel computing. Maze puzzle is one of the games that requires different complex designs. In this project Kruskal's algorithm is used to generate random maze designs and their corresponding solutions. To generate and solve random mazes,a parallel software implementation of maze generator and solver using Kruskal's algorithm is implemented. Our analysis shows that the proposed system partitions the task in a balanced way so that each core has the same amount of job to do. The study reveals that the proposed system offers high performance and CPU utilization. It has been observed that the computation time taken by parallelized maze generator and solver is significantly less than sequential maze generator and solver. CPU is efficiently used when parallely executed.

KEYWORDS: Multi-core; Kruskal algorithm; Maze generation; CPU utilization;

I. INTRODUCTION

Time is the most influential factor in this world. In the recent era, time is the major constraint. Now-a-days people expect the existing or the emerging technologies should produce the effective result in a quicker time. This project mainly concentrates on time reduction and effective CPU utilization. It parallelizes a program that automatically generates and solves mazes in a reduced time with effectively using CPU. Each time the program is run; it will generate and print a new random maze and solution. This program uses the disjoint set (union-find) data structure implemented in Kruskal's algorithm. Parallelization is applied to this algorithm in order to get a better result.

Parallelization is the process used for reducing the time that provides efficient result for the real time applications. It makes programs run faster because this process simultaneously uses more than one CPU or processor. The term parallel processing is used to represent a large class of techniques which are used to provide simultaneous data processing tasks for the purpose of increasing the computational speed of a computer system. In this project, OpenMP tool is used for parallelization process. There are different applications that supports parallelization concept For many real world applications, time is very important parameter mainly in weather forecastng, image processing, flood forecasting, financial prediction and also in many gaming applications.

Multicore architectures are replacing single-core architectures at a rapid pace. Indeed, multicore architectures offer better performance and energy efficiency for the same silicon footprint. As a result, multicore architechures can now be found in a wide range of system architectures, including servers, desktops and embedded and portable architectures. The move towards chip-level multiprocessing architectures with a large number of cores continues to offer dramatically increased performance and power characteristics.

A major challenge in the transition of single-core to multicore architectures is the parallel software development. Most of the software applications are developed following the sequential execution model, which is naturally implemented on traditional single-core processors. Therefore, each new, more efficient, generation of single-core processors translates into a performance increase of the available sequential applications. However, the current stalling of clock frequencies prevents further performance improvements. In this sense, it has been said that sequential programming is dead. Thus, in the current scenario, it is not wise to rely on more efficient cores to improve performance but in the appropriate coordinate use of several cores, that is, concurrency. So, the applications that can benefit from performance increases with each generation of new multi-core and many-core processors are the parallel ones.

A parallel version of a sequential program will run N times faster on N processors, yielding a speedup of N. Parallel computing can increase the performance of the applications by executing them on multiple processors. The applications

have to be programmed in the most efficient way to exploit parallelism. Developing a parallel application involves the portioning of the workload into tasks, and the mapping of the tasks into workers.
The rest of this paper is structured as follows. Section II discusses literature survey. System model is explained in Section III. Experimental setup and results was given in Section IV. Conclusion and future enhancement was given in Section V.

II. LITERATURE REVIEW

Parallelization is required in many gaming applications. One such gaming application is maze generation. Christian Grimme et al [1] proposed a predator–prey model (PPM) that analyzes the architecture and classifies its components with respect to a recent taxonomy for parallel multi-objective evolutionary algorithms, theoretically examined the benefits of simultaneous predator collaboration on a spatial population structure and give insights into solution emergence. The model integrates a gradient-based local search mechanism to exploit problem independent parallelization and hybridize the model in order to achieve faster convergence and solution stability, thus a good approximation is achieved. Parallel computers for dynamic multi-objective optimization problems (DMO) using evolutionary algorithms were designed by Mario Camara et al [2]. In DMO problems, the objective functions, the constraints, and hence, also the solutions, can change over time and usually demand to be solved online. Thus, high performance computing approaches, such as parallel processing, should be applied to these problems to meet the quality requirements within the given time constraints. Two generic parallel frameworks were designed for multi-objective evolutionary algorithms. These frameworks are used to compare the parallel processing performance of some multi-objective optimization evolutionary algorithms. Nick Sephton et al [3] present a study of the relative effectiveness of different types of parallelization, reduction in time for the generation of graphs. Paper was concluded that, worth noting of tree parallelization is less effective when applied to ISMCTS, which is somewhat surprising. Using tree parallelization in an environment where agents are unlikely to be blocked should increase efficiency, but the relative decrease in efficiency suggests that more blocking is occurring. Sanjay Kumar Sharma [5] proposes parallel algorithms for computing the solution of system of linear equations and approximate value of π is presented. The parallel performance of numerical algorithms on multicore system has been analyzed and presented. The experimental results reveal that the performances of parallel algorithms are better than sequential. The parallel algorithms using multithreading features of OpenMP are implemented. Tien-Hsiung Weng [6] proposes two parallel algorithms for sparse matrix transposition and vector multiplication using CSR format: with and without actual matrix transposition. Both algorithms are parallelized using OpenMP. Experimentations are run on a quad-core Intel Xeon64 CPU E5507. This paper measures and compares the performance of these algorithms with that of using CSB scheme. This experimental results show that actual matrix transposition algorithm is comparable to the CSB-based algorithm; on the other hand, direct sparse matrix-transpose-vector multiplication using CSR significantly outperforms CSB-based algorithm. Rohit Chandra et al [7] published a book on Parallel Programming in OpenMP. The goal of this book is to minimize the complexities introduced when adding parallelism to application code. This book is based entirely on the OpenMP efforts in different languages like C, C++ and FORTRAN. Vladimir Loncar et al [8] consider large graphs that can not fit into memory of one process. Experimental results show that Prim's algorithm is a good choice for dense graphs while Kruskal's algorithm is better for sparse ones. Poor scalability of Prim's algorithm comes from its high communication cost while Kruskal's algorithm showed much better scaling to larger number of processes. A variety of classical finite graph algorithms were introduced by Dana Vrajitoru et al [9] together with an analysis of their complexity. Finite graph algorithms starts with the formal definition and classification of graphs. It then discussed implementation details and a C++ graph class example. Breadth-first and depth-first traversals are discussed and an application to counting the connected components of a graph is provided with Dijkstra's algorithm , followed by Kruskal's and Prim's. The problems of topological sorting and maximum flow are discussed. Finally, special tours such as Hamiltonian are discussed, as well as their feasibility. Many of the algorithms in graph theory are named for the mathematician that developed or discovered by Kent D. Lee et al [10]. Dijkstra and Kruskal are two such mathematicians and this chapter covers algorithms developed by them. Representing a graph can be done one of several different ways. The correct way to represent a graph depends on the algorithm being implemented. Graph theory problems include graph coloring, finding a path between two states or nodes in a graph, or finding a shortest path through a graph among many others. There are many algorithms that have come from the study of graphs. To understand the formulation of these problems it is good to learn a little graph notation. Samidh Chatterjee et al [11] proposes GEOFILTERKRUSKAL, an algorithm that computes the minimum spanning tree of P using well separated pair decomposition in combination with a simple modification of Kruskal's algorithm. When P is sampled from uniform random distribution, we show that our algorithm takes one parallel sort plus a linear number of additional steps, with high probability, to compute the minimum spanning tree. Experiments shows that the algorithm works better in practice for most data distributions compared to the current state of the art . It is easy to parallelize and to our knowledge, is currently the best practical algorithm on multi-core machines for $d > 2$. Wen-Qi Zhang et al [12] analyzed several characteristics of four types of cross cells and presenting three cycle-merging operations and one extension operation, proposed an algorithm that finds in an n-vertex grid graph G, a long path of length at least 5/6n+2. This approximation algorithm runs in quadratic time. In addition to its theoretical value, this work has also an interesting application in image-guided maze generation the found path is sufficiently long, a practical application of maze design becomes possible, showing that the pixels in a guided image are almost completely gone through by the

path. A new approach to the procedural generation of maze-like game level layouts by evolving CA(Cellular Automata) was introduced by Andrew Petch et al [13] . The approach uses a GA(Genetic Algorithms) to evolve CA rules which, when applied to a maze configuration, it produces level layouts with desired maze-like properties. The advantages of this technique is that once a CA rule set has been evolved, it can quickly generate varying instances of maze-like level layouts with similar properties in real time. Todd W.Neller et al [14] defines the Rook Jumping Maze, provide historical perspective, and describe a generation method for such mazes. When applying stochastic local search algorithms to maze design, most creative effort concerns the definition of an objective function that rates maze quality defines several maze features to consider in such a function definition. Finally, share the preferred design choices, make design process observations, and note the applicability of these techniques to variations of the Rook Jumping Maze. In the existing system, generation of maze designs and their corresponding solutions are tedious process. It takes more time for generating and solving a maze.Various designs are not implemented in the existing system. Generation of maze puzzles with different designs, different complexities and their corresponding solutions at a very faster rate is not possible in existing system. Many schools during competitions used to search through internet for maze images but there not able to get different designs and the solutions for the same. They have to solve the maze manually for results, it takes more time for the organizers to solve. So they are facing major problems in declaring the results while conducting summer camps, competitions, parent-teachers games.

III. SYSTEM MODEL

Overall system design is given in Fig. 1. In this project, a program for generating and solving a maze is parallelized using OpenMP. This involves different vatiety of designs and their corresponding solutions using Kruskal's algorithm. Using this project one can easily generate different mazes and also organizers can attain the results very quickly. In this project maze generation is developed using Kruskal's algorithm because it is the best recursive back tracker. After generating a maze, this program then solves the maze using a modified pre-order traversal. The search algorithm will begin at the start room and search for the finish room by traversing wall openings. The search should terminate as soon as the finish room is found. This program outputs the order in which rooms where visited and indicates the shortest solution path from start to finish.

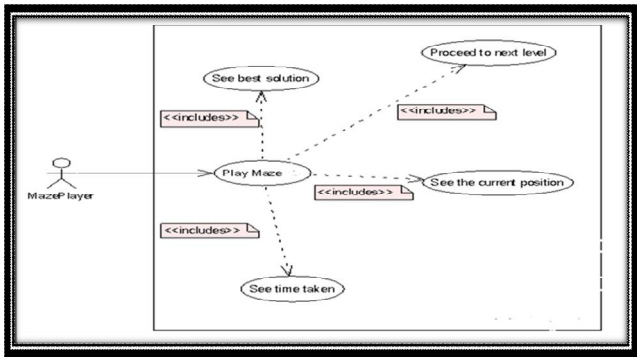

Fig. 1 System model

Maze generator system is designed in such a way that it generates different maze designs for each input sets. It provides different complexities in maze design.It allows the maze player to see the best solution, to track the current position. The maze player can also note the time for maze generation as well as for solving the maze. The maze player can track different solutions for one design using this system. Tracking the current position helps the maze player to proceed the game from being struck at a particular point. But for the above mentioned process it takes more time so the maze generator is parallelized using OpenMP. Parallelization is used for reducing the time. It makes program run faster because this process simultaneously uses more than one core or processor. Running more than one core leads to quicker generation of maze and its corresponding solution. The first module is to develop Kruskal's algorithm in C++. Kruskal's algorithm is the best recursive back tracker. Kruskal's algorithm is developed to generate and solve maze.It starts with the set of all vertices and an empty set of edges. On each iteration an edge is added to the collection. The only restriction in this process is that the addition of an edge should not create a circuit, a path that starts and ends with the same vertex . The second module is to develop maze generation, it needs a data structure to represent the maze of rooms and walls. Since the size of the graph is known at startup, a 2-dimensional array-based implementation that mimics the room grid structure may work well. It requires the player to trace a path from one point, the start, to another, the end, on a 2-dimensional grid. This grid contains white space, referred to here as a "path", on which the player must travel, and black space, referred to here as a "wall", which the player cannot cross. This process will require two additional data structures. First, need an efficient way to randomly select walls, such that a single wall is never selected more than once. Second, need an efficient way to test if the selected wall is allowed to be removed from the maze (according to rule #2 above). This must use the disjoint set data structure. The third module is to parallelize the maze generator system. Parallelization process is applied using OpenMP tool for the maze generator and solver. This process simultaneously uses more than one CPU cores. The parallel processing is used to represent a large class of techniques which are used to provide simultaneous data processing tasks for the purpose of increasing the computational speed of

a computer system.It is implemented in C++.The directive omp.h has been included inorder to achieve parallelization.For loop has been parallelized by adding OpenMP directive #pragma omp parallel for default(shared) private() reduction().In addition to this, loop unrolling has been performed to parallelize further. Loop unwinding, also known as loop unrolling, is a loop transformation technique that attempts to optimize a program's execution speed at the expense of its binary size(space time tradeoff).The transformation can be undertaken manually by the programmer or by an optimizing compiler. The goal of loop unrolling is to increase a program's speed by reducing(or eliminating)instructions that control the loop,such as pointer arithmetic and "end of loop" tests on each iteration;reducing branch penalties; as well as "hiding latencies, in particular, the delay in reading the data from memory".To eliminate this overhead, loops can be re-written as a repeated sequences of similar independent statements. Parallelization is applied to the proposed system offers that high performance,scalability over different number of cores.It has been observed that the computation time taken by parallelized maze generator and solver is significantly less than sequential maze generator and solver. Testing is done for different sets of input, it involves the grid size of about 1000*1000 matrix . For each set of input, the maze and its corresponding solution is generated within a minimum time .It also measures the CPU performance. For each maze generation the CPU performance is measured. For each set of input random mazes are designed.

IV. Experimental Setup and Results

The program is developed in C++ and run in visual studio 2010. For parallelization OpenMP API is used which requires omp.h. The program is run for different maze sizes. The maze sizes are randomly taken. The minimum value 5*5 cm and maximum value is 19*800 cm. As the generated mazes are to be printed and utilized, the maximum size is fixed as 19*800. If the size of the maze exceed this value then it is not possible to visualized the entire maze. Each maze is given a input size. For each maze that are specified with the size is solved serially as well as parallel to find the solution in a reduced time. The following are the test cases that had been undergone. 7*8 maze is generated and then it is solved parallel.

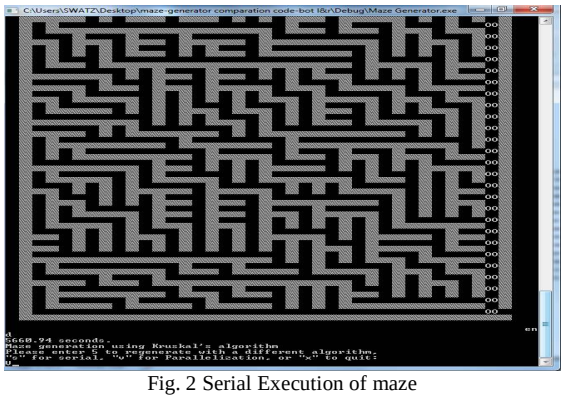

Fig. 2 Serial Execution of maze

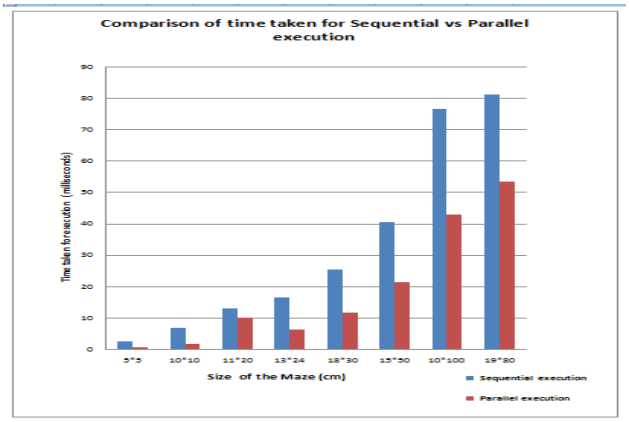

Fig. 4 Comparison of time taken for sequential vs parallel execution

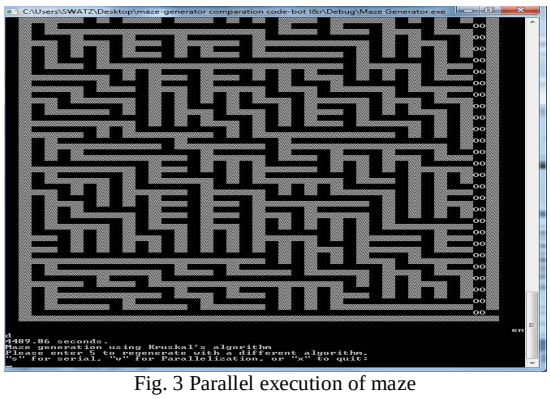

Fig. 3 Parallel execution of maze

Maze generator is designed to generate random mazes. For each set of input it generates different maze designs. From the above table it is shown that the maze is specified with different sizes, 5*5 maze is generated in 0.197 milliseconds, 10*10 maze is generated in 0.143 milliseconds,11*20 maze is generated in 0.155 milliseconds, 13*24 maze is generated in 0.164 milliseconds, 18*30 maze is generated in 0.377 and 15*50 maze is generated in 0.352 milliseconds. The size of the maze should not exceed 19*1000 because the size of the output screen is the same.If the value exceeds 19*1000 then it is not possible to visualize the entire maze. The comparison is made between sequential and parallel solving of maze. The sequential execution of maze solution takes more time when compared to parallel execution of maze.5*5 maze- when solved parallely, it is 1.828 milliseconds less than the serial execution.10*10 maze- when

solved parallely, it is recorded that about 5.37 milliseconds less than the serial execution.11*20 maze- when solved parallely, it is noted that about 2.797 milliseconds less than the serial execution. Fig. 2 shows serial maze execution. 13*24 maze- when solved parallely, it is observed that about 10.23 milliseconds less than the serial execution.18*30 maze- when solved parallely, it is 13.571 milliseconds less than the serial execution .15*50 maze- when solved parallel, it is obtained that19.167 milliseconds less than the serial execution. Fig. 3 shows parallel execution of maze.

When the size of the maze increases from minimum size to maximum size the more time is required to solve the maze, so parallelization is applied to reduce the time. This is observed interms of percentage .As shown in the fig 6.25,when 5*5 maze is solved parallely it is observed that about 23% of the time is increased compared to serial execution.Then when 10*10 maze solved parallel it is noted that about 21% of the time is increased,considering 11*20 maze it is recorded that about 78% of the time is increased compared to serial execution. When 13*24 maze is solved parallely it is observed that about 37% of the time is increased compared to serial execution. When 18*30 maze is solved parallely it is observed that about 46% of the time is increased compared to serial execution. Considering 15*50 maze it is recorded that about 52% of the time is increased compared to serial execution. When 10*100 maze is solved parallely it is observed that about 56% of the time is increased compared to serial execution. Considering 19*80 maze it is recorded that about 65.9% of the time is increased compared to serial execution.

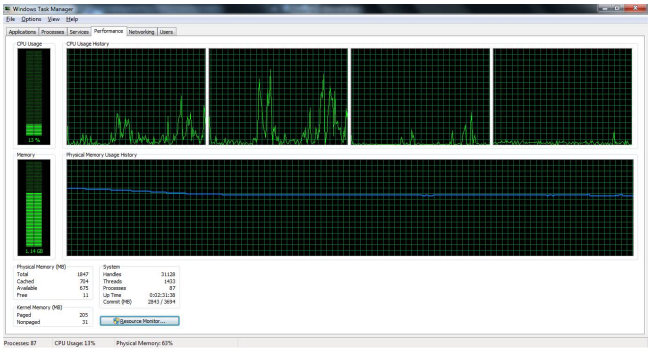

Fig. 5 CPU utilization of serial execution

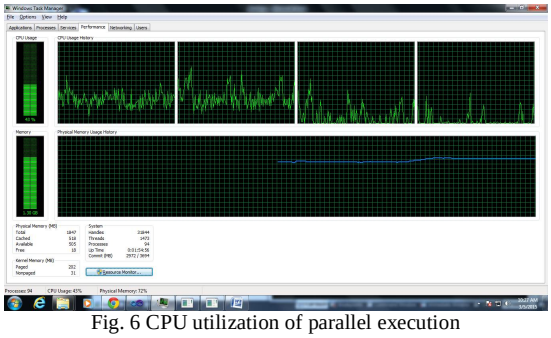

Fig. 6 CPU utilization of parallel execution

Different values for the maze are specified and their corresponding CPU performance is measured For 19*800 maze, CPU utilization is recorded using task manager. The corresponding performance measure is shown in the Fig 5 and Fig 6. When parallel execution is done about 30% of CPU utilization is increased than that of serial execution.

V. CONCLUSION

To generate and solve random mazes,a parallel software implementation of maze generator and solver using Kruskal's algorithm is implemented.Our analysis shows that the proposed system partitions the task in a balanced way so that each core has the same amount of job to do.The study reveals that the proposed system offers high performance and CPU utilization.It has been observed that the computation time taken by parallelized maze generator and solver is significantly less than sequential maze generator and solver. CPU is efficiently used when parallely executed.

REFERENCES

[1] Christian Grimme, Joachim Lepping, Alexander Papaspyrou(2012) 'Parallel predator–prey interaction for evolutionary multi-objective optimization'- Natural Computing Vol.11, pp.519-533.
[2] Mario Cámara, Julio Ortega(2012) 'Comparison of Frameworks for Parallel Multiobjective Evolutionary Optimization in Dynamic Problems' , pp.101-123.
[3] Nick Sephton, Peter I. Cowling, Edward Powley and Daniel Whitehouse ' Parallelization of Information Set Monte Carlo Tree Search' York Centre for Complex Systems Analysis, Department of Computer Science, University of York, United Kingdom.
[4] Carolina García-Costa 'Speeding Up the Evaluation of a Mathematical Model for VANETs Using OpenMP' -AISC-197, pp.23-37.

[5] Sanjay Kumar Sharma, Kusum Gupta(2013) 'Parallel Performance of Numerical Algorithms on Multi-core System Using OpenMP' - International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol.2.
[6] Tien-Hsiung Weng, Delgerdalai Batjargal (2013) 'Parallel Matrix Transposition and Vector Multiplication Using OpenMP' Vol.234, pp. 243-249.
[7] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, Ramesh Menon ' Parallel Programming in OpenMP' ch.1-6, pp.1-249.
[8] Vladimir Lončar, Srdjan Škrbić, Antun Balaž(2014) 'Parallelization of Minimum Spanning Tree Algorithms Using Distributed Memory Architectures' -Transactions on Engineering Technologies , pp.543-554.
[9] Dana Vrajitoru, William Knight(2014), 'Finite Graph Algorithms'- Practical Analysis of Algorithms pp.361-452.
[10] Kent D. Lee, Steve Hubbard(2015) 'Graphs' -Data Structures and Algorithms with Python .
[11] Samidh Chatterjee, Michael Connor, Piyush Kumar(2010) 'Geometric Minimum Spanning Trees with GEOFILTERKRUSKAL'- Experimental Algorithms , SEA-2010, LNCS-6049, pp.486–500.
[12] Wen-Qi Zhang, Yong-Jin Liu(2011) 'Approximating the longest paths in grid graphs'- Theoretical Computer Science, Vol.412,pp 5340–5350.
[13] Andrew Pech, Philip Hingston, Martin Masek(2015) 'Evolving Cellular Automata for Maze Generation'- in Artificial Life and Computational Intelligence, Vol.8955, pp.112-124.
[14] Todd W. Neller, Adrian Fisher, Munyaradzi T. Choga, Samir M. Lalvani, Kyle D. McCarty ' Rook Jumping Maze Design Considerations'- Vol.6515, 2011, pp.188-198.