Journal of Theoretical and Applied Information Technology
15th October 2018. Vol.96. No 19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

COMPARATIVE STUDY OF PARALLEL IMPLEMENTATION FOR SEARCHING ALGORITHMS WITH OPENMP

[1]RENEA CHOWDHURY SHORMEE, [2]RAVIE CHANDREN MUNIYANDI, [3]DIP NANDI

[1]Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Malaysia

[2]Center for Cyber Security, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Malaysia

[3]Department of Computer Science, American International University Bangladesh, Bangladesh

E-mail: [1]reneachowdhury@gmail.com, [2]ravie@ukm.edu.my, [3]dip.nandi@aiub.edu

ABSTRACT

Investigating the multi-core architecture is an essential issue to get superior in parallel reenactments. However, the simulation highlights must fit on parallel programming model to build the execution. The main goal of this research is to choose and evaluate parallelism using OpenMP over sequential program. For this purpose, there is a portrayal of two searching algorithms. The calculation is to discover the next edge of Prim's algorithm and single source shortest way of Dijkstra's algorithm. These two algorithm actualized in sequential formulation. Parallel searching algorithms are then implemented in view of multicore processor. The speed-up ratio and efficiency of parallel searching algorithms are tested and investigated in SGEMM GPU Kernel performance dataset with 241600 records and 18 attributes. Results show the dataset with different data sizes achieved super linear speed-up ratio and efficiency on OpenMP by running on 4 cores processor and reduction of the running time over sequential program. More importantly, the new implementation drastically decreases the time of execution for thread 8 for Prims algorithm from 5.16ms to 1.48 ms for Dijkstra algorithm. Parallel calculation is impressively powerful for huge graph size. General outcome shows that multi-threaded parallelism is exceptionally successful to accomplish better performance for dataset based on searching algorithms by separating the primary dataset into sub-datasets to increase diversity on arrangement investigation.

Keywords: *Sequential programming, Parallel programming, Prim's, Dijkstra's, OpenMP*

1. INTRODUCTION

Finding the shortest distance for all objects in a graph is a common task in solving many day to day and logical issues. The algorithm for finding the shortest path, discover their application in numerous fields, for example: Google maps, routing protocol and so on. There are two algorithms for finding the nearest way and single source shortest path, utilizing two algorithms Prims algorithm [4-5] and Dijkstra's algorithm [1-3]. To improve the searching, the best use of shortest path is to implement the parallelism.

With the rapid improvement of urban communities, congested road turned into a concerning issue. Along these lines the Intelligent Traffic System is developing rapidly and the shortest path optimization is an important part of this problem. This issue has been the research hotspot for long time and for the sequential shortest path optimization, individuals have gotten many research comes about and applied in many applications [1].

Big data mostly comes from people's day-to-day activities and Internet-based companies. Big data represents content and cloud computing is an environment that can be used to perform tasks on big data. Nonetheless, the two concepts are connected. In fact, big data can be processed, analyzed, and managed on cloud. Parallel algorithms can be implemented in the cloud-computing environment to reduce computation time, memory usage and I/O overhead for generating frequent item sets [14].

Moreover, several single source shortest path algorithms and minimum spanning tree have been computed in order to resolve this issue, to saves time using parallel process to quickly focus only on the results of attentiveness. For this study, the parallel computation is a proficient method to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

enhance the greedy algorithms containing large data.

This research aim to achieve the following objectives:

i) To propose and implement the programming process between sequential and parallel programming that required the less execution time for large datasets.
ii) To investigate and evaluate the performance of parallel process over sequential using OpenMP over sequential programming of two searching algorithms.

In sequential algorithm implementation, it requires long time to discover the shortest distance if all sets of vertices are in the graph. So it is troublesome assignment to locate the most nearest node from source to goal. Both of the searching algorithm problem starts initially considering source as A to all other vertices in datasets. The graph solving problems increment in size, effective parallel shortest path handling becomes important as computational and memory prerequisites increment [2]. For large structure or framework it requires long time to perform their tasks and this is the reason why the parallelization used to perform operation in less time. Execution for the task in less time to diminish the proficiency and speedup factor utilizing the OpenMP and furthermore utilize the parallel Prims algorithm, parallel Dijkstra algorithm [1,6-8]. Multicore processors can likewise execute various assignments at a single time [3].

The next section of this study, Section 2 presents a literature review on the research work. Section 3 introduces methodology of parallelization framework for OpenMP. Section 4 elaborates on the experiments and findings from the experiments. Finally, Section 5 concludes this research based on the investigation.

2. LITERATURE REVIEW

Cao et al. [1] proposed the plan to apply the non hierarchical algorithm such as Dijkstra's algorithm to various levels and the entry and exit points (node E) between a high-level and low-level are acquired by the heuristic coordinating search approach.

According to Awari [2] graph problem solve by utilizing the standard graph Algorithm. There is a depiction of two algorithm, Floyd War shall algorithm and Dijkstra's algorithm. Parallel algorithm is impressively compelling for large graph size. These two algorithm execute in serial formulation. Parallel algorithm utilized for calculating or finding shortest way of graph. With the help of graph algorithm these tasks should be possible in parallel and reduce the computation time and efficiency.

In Pathare & Kulkarni [3] the parallel technique was applied to achieve the best performance by reducing the execution time using OpenMP for matrix multiplication algorithm along with floyd war shall algorithm on a single processor or multi core processor for a sequential execution and then obtained the good performance by parallelization.

The key algorithm of parallel innovation in view of OpenMP is progressed to arrange the promoter data in Shi et al. [8]. The promoter data are incepted from the upstream region of five plants successions. The preparation integrates disturbing the arrangements in random, figuring the P esteem, choosing the noteworthy themes from the continuous themes, marking the arrangement number and so on. The serial algorithm and the parallel algorithm are analyzed in the usage system.

In Anjaneyulu et al. [10] proposed algorithms produces the spanning tree using Prim's algorithms for parallelism, usefulness of Prims algorithms comes when there are more number of edges into the graph and it performs faster. With regards to speed and time complexity the serial algorithm takes more time to execute and not an achievable solution for the problem for getting approximation for metric travelling salesman problem (TSP). In the way of getting the minimum spanning tree, the tree traversed using depth first traversal in which use of parallelism and recursion was provided.

Jasika et al. [11] presents the issue of parallelization of Dijkstra's algorithm. Here Dijkstra's shortest path algorithm is actualized and displayed, and the exhibitions of its parallel and serial execution are looked at. The algorithm execution was parallelized utilizing OpenMP and OpenCL techniques. Its performance were measured on 4 different configurations. The results demonstrate that the parallel execution of the algorithm has great performances as far as terms of speed-up ratio, when compared to its serial execution.

In Maroosi et al. [12], an inventive classification algorithm based on a weighted system is presented. Two new algorithms have been proposed for recreating membrane frameworks models on a Graphics Processing Unit (GPU). Correspondence and synchronization amongst strings and string hinders in a GPU are tedious procedures. In past examinations, subordinate items were appointed to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

various strings. This builds the requirement for correspondence amongst strings, and accordingly, execution diminishes.

In Ang et al. [13], to acquire the great execution for CVS on multi-core systems, it is vital to use parallelism devices proficiently. These parallelism devices should be used on hotspots so as to limit improvement time, to diminish application advancement costs. This is a testing task and requires an inside and out examination of multi-core systems.

The Apriori algorithm is extraordinary compared to other classical algorithms for finding continuous item sets from a value-based database, yet it has a few disadvantages, for example, that it scans the dataset commonly to produce frequent item sets and that it creates numerous candidate item sets. At the point when data mining mainly deals with large volumes of data, both memory utilization and computational cost can be very high, additionally, a single processor's memory and central processing unit resources are limited, which impacts the inefficient execution of the algorithm. In Saabith et al. [14] Apriori algorithm has numerous downsides for preparing tremendous datasets. Parallel and distributed computing used for the better solution for conquer the above issues.

The approach used in Maroosi et al. [15] is the parallel membrane computing model to execute parallelized harmony search efficiently on different cores, where the film figuring correspondence attributes were utilized to trade data between the activities on various cores, in this way expanding the decent variety of congruity look and enhancing the execution of concordance seek.

In Maroosi & Muniyandi [16], another model of membrane computing with dynamic films is characterized for taking care of the N-queens issue. This model builds the parallelism of past Membrane registering with dynamic membranes. Correspondence rules reduce speed on multi-core processing meanwhile correspondences. Synchronizations between threads and cores that are fundamental for correspondence rules are exceptionally time consuming process.

The related works for the proposed study has been described in this section 2. There are already many research that has been done on this two algorithms for sequential process. Parallel implementation using OpenMP also has been done by researchers. From the section 2, it can been seen that the comparative study of two MST has been done. Also, the analysis has been done on SSSP with matrix multiplication and various algorithms. There is no research on Dijkstra's algorithm with the Prim's algorithm. So, this study proposed the parallel implementation of two algorithms based on two searching algorithms which are greedy algorithms using OpenMP, where the Prim's algorithm is the optimal solution of MST and Dijkstra's algorithm is the optimal solution of SSSP of greedy algorithm.

3. METHODOLOGY

This section introduces a brief background of the two searching algorithms and the parallel framework as OpenMP that was implemented in this research. Both of the searching algorithms are greedy algorithms but have some different criteria. The primary distinction between the two is the rule that is utilized to pick the following vertex for the tree.

3.1 Prim's Algorithm

The algorithm was found in 1930 by mathematician Vojtech Jarnik and later independently by computer scientist Robert C. Prim in 1957. It begins with an empty spanning tree. The idea is to keep up two sets of vertices. The first set contains the vertices already included into the MST, the other set contains the vertices not yet included [4-5]. At each progression, it consider all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other end point of the edge to the set containing minimum spanning tree.

3.2 Dijkstra's Algorithm

Dijkstra's algorithm is used to locate the single source shortest path issue. This solution is a prime example of a greedy algorithm. Considering the single vertex to all different vertices in the graph. Dijkstra's algorithm solves the single source shortest-path problem on directed and undirected graphs with non-negative weights edges. Dijkstra's algorithm find the minimum distance from single vertex [4].Dijkstra's algorithm is like Prim's algorithm. This algorithm fabricates the tree outward from source in a greedy design. Taking after Prim's algorithm, it incrementally find the shortest ways from source to alternate vertices of graph.

3.3 Open Multi-Processing (OpenMP)

OpenMP was introduced in 1997 to standardize programming expansions for shared memory machines. OpenMP is an application worked with the hybrid model of parallel programming which can keep running on a computer cluster using

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OpenMP, to such an extent that OpenMP is utilized for parallelism inside a (multi-core) hub [6]. OpenMP is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most stages, guideline set structures and working frameworks, including Solaris, Linux, macOS, and Windows [7]. There are three components of API such as compiler directive, runtime library routines and environment variables. This directives are represented with "#pragma omp" that help users explicitly build parallelism using constructs. The main technique used to parallelize code in OpenMP are the compiler directives. The directives are added to the source code as an indicator to the compiler of the presence of a region to be executed in parallel, along with some instruction on how that region is to be parallelized.

OpenMP was initially designed to parallelize loop-based sequential programs based on a fork-join model. The model allows one master thread to perform tasks throughout the whole program and forks off threads to process parts of the program that needed to run in parallel. It is simple, portable and extensible [7,9].

3.3.1 OpenMP analysis

There are four scheduling policies accessible in OpenMP: static scheduling, dynamic scheduling, guided scheduling, and runtime scheduling. Static scheduling is to averagely separate the loop iterations and assign them to the individual threads and diminish the collision when visiting shared memory. This research uses static scheduling policy. Dynamic scheduling is not the same as static scheduling by dividing the blocks and utilizing FIFO to handle them. The number of iteration of each time equals to the assigned block size by the schedule clause. At the point when a thread completes the iteration assigned to it, it will request the following group of iteration until the number of iteration is less than the block size. The static scheduling is relatively convenient and also able to improve the efficiency and performance of algorithm [1].

3.3.2 Parallel execution

The algorithm itself is broken into two stages. This is vital in light of the fact that synchronization isn't feasible outside of neighborhood work-bunches in OpenMP and this would be required to execute the algorithm initially in a single thread. The first phase of the algorithm visits all vertices that have been checked and decides the cost to each neighbor [8]. The second period of the algorithm verifies whether a smaller cost has been found for every vertex and provided that this is true, marks it as requiring appearance and updates the cost. At the end of a parallel region, the thread teams are stopped and the master thread continues execution [11].

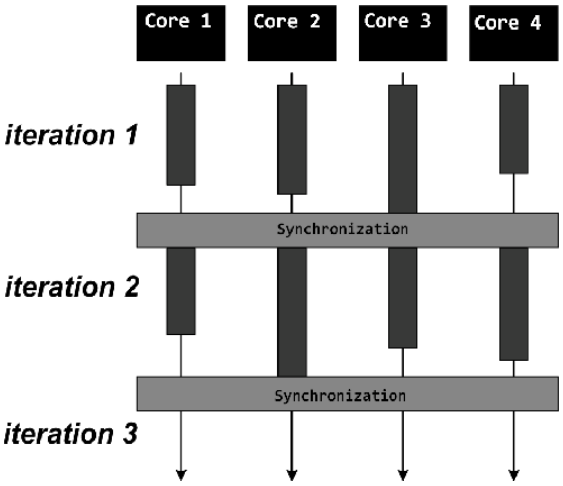

Figure 1: Iteration During Execution

Here figure 1 illustrates the iteration process during the execution of parallelized Prim's and Dijkstra's algorithms. For this proposed work the hardware configuration consists four processors as a multicore processor [1].

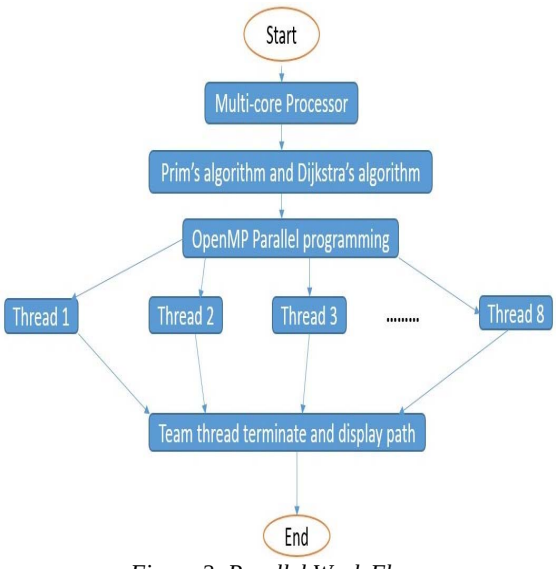

Figure 2: Parallel Work Flow

Figure 2 shows the work flow for both algorithms using OpenMP techniques and threads.

4. PERFORMANCE ANALYSIS

In this chapter, the simulation is carried out to estimate the performance of the proposed parallel implementation, which represents the comparison

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of speedup and efficiency for two algorithms. Moreover, OpenMP is extensively used. The performance will be using performance metrics with dataset for parallel program based on different number of threads.

4.1 System Configuration

In this section, the hardware and software configuration are described as follows:

4.1.1 Hardware configuration

1) Processor: Intel Core i5-5200U CPU @ 2.20GHz 2.19GHz
2) Memory: 4GB

4.1.2 Software configuration

1) Operating system: Windows version 8.1
2) Compiler and code editor: Dev C++ version 5.11 TDM-GCC
3) Communication protocol: OpenMP

4.2 Dataset

The dataset used for testing was the SGEMM GPU Kernel performance dataset which was taken from the UCI Machine Learning repository [20]. The informational collection contains 241600 records with each record having eighteen attributes, which are all integer. The records are divided into 5 classes as 50k, 100k, 150k, 200k and the SGEMM dataset containing all records.

4.3 Performance Metrics

Subsections beneath describe the execution analysis metrics used to assess the proposed algorithms.

4.3.1 Execution time

While invoking each algorithm, it start to check the time and the precision is milliseconds. At the point when the calculations are done, which implies the shortest paths have been discovered, the counter is halted and the execution time is shown on the screen [7, 13].

4.3.2 Speedup ratio

Equation (1) is used to measure the execution time performance gain [7, 13] in parallel computation analysis,

$$S = \frac{T_{serial}}{T_{parallel}} \quad (1)$$

where T_{serial} is the best serial code execution time, divided by parallel code execution time $T_{parallel}$ solving the same problem with p processors or threads [7].

4.3.3 Efficiency

Correlated to speedup, to measure the parallel performance on average percentage of non-idle time,

$$E(\%) = \frac{S}{p} = \frac{\left(\frac{T_{serial}}{T_{parallel}}\right)}{p} = \frac{T_{serial}}{p.T_{parallel}} .100 \quad (2)$$

where S is the speedup defined in Equation (1), and p is the number of processors or threads, essentially $T_{parallel}$, S and E are depend on p. Efficiency is denoted in percentage (%) [7, 13].

4.4 Results

This section represents the comparison and analysis of performance of two proposed algorithms. Each has two versions: sequential and parallel, thus this study can compare and analyze the performance. Both of the programs are executed on same processor machine. In table 1 and 2, the execution time for sequential and parallel program are recorded to compare the results of sequential vs parallel. Execution time is recorded against different dataset to analyze the speedup of parallel algorithm against sequential [3].

Table 1: Sequential And Parallel Contrast Time of Prim's Algorithm for Different Threads

Datasets	Sequential Prim's	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
50k data	1.537000	12.59	6.55	5.00	2.48	1.97	4.72	5.04	5.04
100k data	1.518000	12.52	6.86	4.64	2.52	2.07	4.66	4.91	5.04
150k data	1.541000	12.59	6.80	3.52	3.06	2.24	3.12	5.01	4.96
200k data	2.149000	12.59	6.63	4.20	2.46	1.98	4.48	4.83	5.08
SGEMM dataset	2.293000	12.58	6.66	3.99	2.40	2.31	5.20	5.46	5.16

Journal of Theoretical and Applied Information Technology
15th October 2018. Vol.96. No 19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TABLE 2: SEQUENTIAL AND Parallel Contrast Time of Dijkstra's Algorithm for Different Threads

Datasets	Sequential Dijkstra's	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
50k data	3.459000	12.60	6.54	4.48	2.54	2.02	1.62	1.47	1.48
100k data	4.086000	12.58	6.56	4.61	2.54	2.01	1.64	1.47	1.48
150k data	4.088000	12.59	6.59	4.58	2.54	2.03	1.79	1.48	1.48
200k data	4.957000	12.63	6.62	4.84	4.24	2.65	2.11	1.90	1.47
SGEMM dataset	5.494000	12.58	6.68	4.54	4.12	2.67	2.08	1.71	1.48

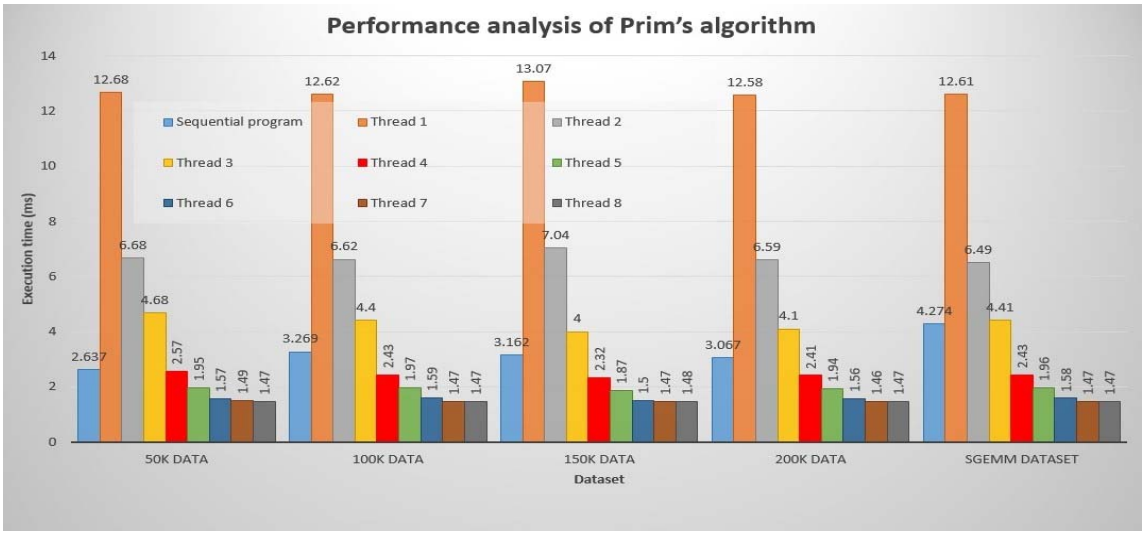

Figure 3: Performance Analysis of Prim's Algorithm for Sequential Program and Different Threads with Different Data Sizes

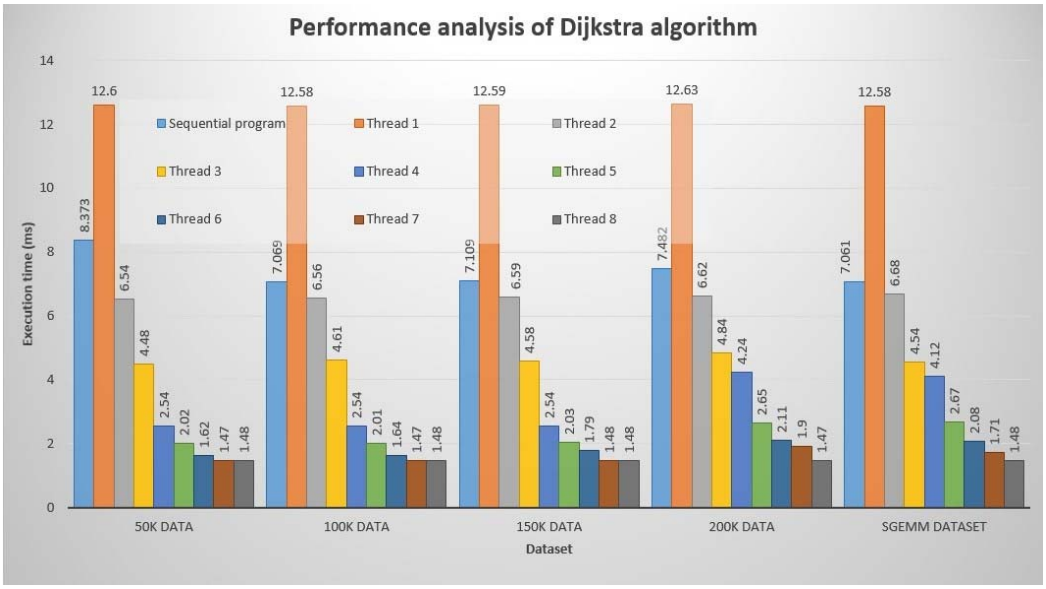

Figure 4: Performance Analysis of Dijkstra's Algorithm for Sequential Program and Parallel Program for Different Threads with Different Data Sizes

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 3: Speed-Up Ratio and Efficiency of Prim's Algorithm for Different Threads

		Number of Threads							
Datasets	**S and E**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
50k	S	0.1221	0.2346	0.3074	0.6197	0.7802	0.3256	0.3049	0.3049
	E	0.1221	0.1173	0.1024	0.1549	0.1560	0.0542	0.0435	0.0381
100k	S	0.1212	0.2212	0.3271	0.6023	0.7333	0.3257	0.3091	0.3011
	E	0.1212	0.1106	0.1090	0.1505	0.1466	0.0542	0.0441	0.0376
150k	S	0.1224	0.2266	0.4377	0.5035	0.6879	0.4939	0.3075	0.3106
	E	0.1224	0.1133	0.1459	0.1258	0.1375	0.0823	0.0439	0.0388
200k	S	0.1706	0.3241	0.5116	0.8735	1.0853	0.4796	0.4449	0.4230
	E	0.1706	0.1620	0.1705	0.2183	0.2170	0.0799	0.0635	0.0528
SGEM M	S	0.1822	0.3442	0.5746	0.9554	0.9926	0.4409	0.4199	0.4443
	E	0.1822	0.1721	0.1915	0.2388	0.1985	0.0734	0.0599	0.0555

Here, figure 3 represents the execution time of sequential and parallel program for Prims algorithm. Then figure 4 shows the same actions for Dijkstra's algorithm with impressive result completed within 1.47ms. From the execution time, calculated the speed-up ratio S for Prim's and Dijkstra's algorithm using equation (1), which are represented in table 3 and 4 respectively. Then it executed equation (2) and get the efficiency E in table 3 and 4. In figure 5 and 6, it can be seen that the efficiency of Prim's and Dijkstra's has been executed. The ratio of efficiency is between 0 and 1. Efficiency varies with different number of threads and data sizes. If the S_p= P, E= 1 then the algorithm is a best parallel algorithm. This is only the ideal situation and in reality it is impossible since the ratio is affected by the low degree of parallelism, lack of load balance and communication collision, etc [1,7,13]. There are significant changes for the results of computation time of algorithms for both techniques.

4.5 Significance

The fundamental significance of this research can be compressed in the stated points:

1) Execution Time: The Sequential searching algorithm and OpenMP searching algorithm programs are infused with time estimation function code to stamp the new best arrangement found on each execution in millisecond granular time to encourage information gathering. Here the running time of Prim's algorithm is 2 times less than Dijksra's algorithm. On the other hand, the running time for OpenMP Prims algorithm is more than 2 times over OpenMP Dijkstra's algorithm.

Table 4: Speed-Up Ratio and Efficiency of Dijkstra's Algorithm for Different Threads

		Number of Threads							
Datasets	**S and E**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
50k	S	0.2745	0.5288	0.7721	1.3618	1.7123	2.1352	2.3531	2.3372
	E	0.2745	0.2644	0.2573	0.3404	0.3424	0.3558	0.3361	0.2921
100k	S	0.3248	0.6228	0.8863	1.6086	2.0328	2.4914	2.7795	2.7608
	E	0.3248	0.3114	0.2954	0.4021	0.4065	0.4152	0.3970	0.3451
150k	S	0.3247	0.6203	0.8925	1.6094	2.0137	2.2837	2.7621	2.7621
	E	0.3247	0.3101	0.2975	0.4023	0.4027	0.3806	0.3452	0.3452
200k	S	0.3934	0.7487	1.0241	1.1694	1.8705	2.3492	2.6089	3.3721
	E	0.3924	0.3743	0.3413	0.2922	0.3741	0.3915	0.3727	0.3727
SGEM M	S	0.4367	0.8224	1.2101	1.3334	2.0576	2.6413	3.2128	3.8419
	E	0.4367	0.4112	0.4033	0.3334	0.4115	0.4402	0.4589	0.4589

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

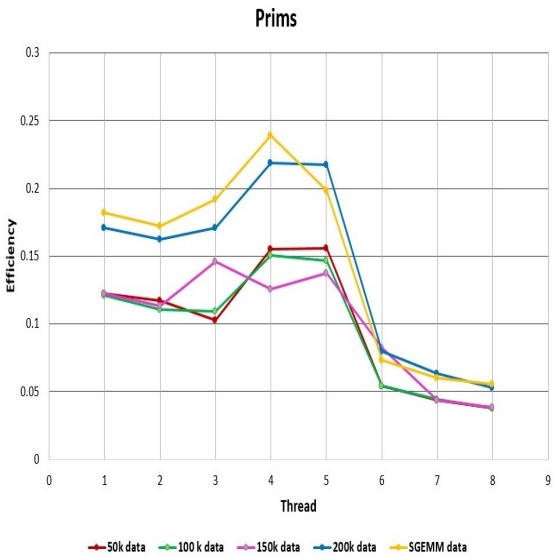

Figure 5: Efficiency of Prim's Algorithm

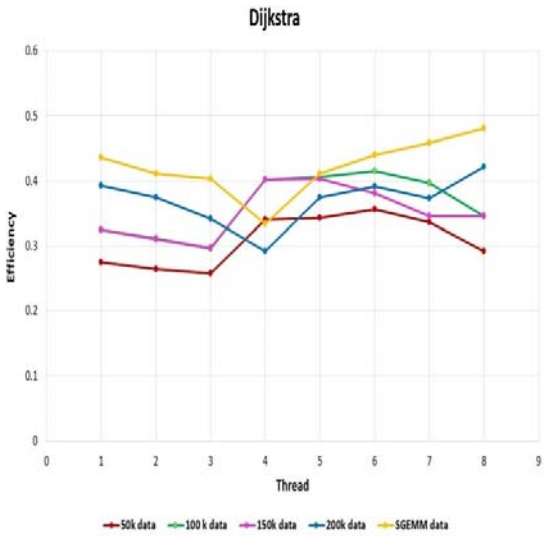

Figure 6: Efficiency of Dijkstra's Algorithm

2) Speed-up ratio: The main goal of this research is to compare the performance. So for that it need to calculate the speed-up ratio from the execution time [1]. Then it can find out the efficiency from speed-up ratio and the numbers of threads used for parallel execution.All the significance of this research is co-related to each other.The speed-up ratio of OpenMP Dijkstra is also more than 2 times over OpenMP Prim's algorithm. The best can hope for is *Tparallel = Tserial/p*. When this happens, then it can be said that the parallel program has linear speedup.For this research it is not linear.

3) Efficiency: The workload of the nodes can be represented as the division between the normal workload and the most extreme workload (considering all nodes). This proportion, in the ratio between 0 and 1, shows how much distinction there is between maximum load and normal, so efficiency close to 1 implies a balanced parallelism,while 0 speaks to the most extreme unbalance [1]. Response time and efficiency are connected, since a low degree of parallelism will introduce worse response time. The two measures are considered to demonstrate the pick-up of the proposed processes from two distinct perspectives. For this research, the efficiency of OpenMP Dijkstra is relatively near to 1 (which is 0.4802) over OpenMP Prim's algorithm (which is 0.2388), which implies a balanced parallelism.

There is a limitation for searching the path from source to destination of Prim's algorithm or the next edge of Dijkstra's algorithm. For the sequential execution the simulation started the searching from node 0 as the source for both algorithms. The source is not chosen randomly by the system. Again, for the parallel computation, the process insert the starting node as 0. Thus it found the next edge according to the edge A as a starting node. This had been done to compare the results further for sequential and parallel program for similar scenario. So, this study only compared the results where the source is nothing but 0 for both algorithms. Also in this study, the experiment has been done by one single machine mentioned in the sub subsection 4.1.1 which could be simulated in different configurations to compare the performance

Main execution bottleneck of Prim's algorithm is correspondence overhead of all-to-one reduce task. Reduce task is costly in contrast with local computation, and every single different process are idle while waiting for diminish to finish. This prevents Prim's algorithm to accomplish significant efficiency on a large number of datasets. Subsequently, Prim's algorithm is best utilized on a smallest graph on which parceled input graph can fit.

5. CONCLUSION

In this study, two searching algorithms are prepared with five sub-datasets by the OpenMP parallel innovation. Initially, the algorithms were sequential. At first, the simulation has been done for sequential program. Then the parallel process of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OpenMP has been applied to get the parallel execution time. This two types of execution time was utilized to compare the performance. The experiment result shows that the parallel algorithms simulated in this research are efficient over the sequential algorithms and the speed-up ratio of the two parallel algorithms are satisfied in finding the shortest path [7]. Also the efficiency of those two parallel algorithms has been compared.

There are few aspects that can enhance the task later on. The conceivable outcomes of different parallel techniques in the programming part and more updated, for instance OpenMP with MPI, OpenMPI, CUDA could be tried. Also, this study intend to test other searching algorithms such as- Kruskal, BFS with required number of threads on the OpenMP based parallelism and determine whether a similar approach can be executed to get better performance of those algorithms in such an environment.

ACKNOWLEDGEMENT

This work is supported by the Fundamental Research Grant Scheme of the Ministry of Higher Education (Malaysia; Grant code: FGRS/1/2015 /ICT04/UKM/02/3).

REFERENCES:

[1] Cao H, Wang F, Fang X, Tu H-L, Shi J. OpenMP parallel optimal path algorithm and its performance analysis. 2009 WRI World Congr Softw Eng WCSE 2009. 2009;1.

[2] Awari R. Parallelization of shortest path algorithm using OpenMP and MPI. Proc Int Conf IoT Soc Mobile, Anal Cloud, I-SMAC 2017. 2017;304–9.

[3] Pathare S, Kulkarni P, Kardel R. Performance Analysis of Algorithm Using OpenMP. 2014;1(1):152–6.

[4] Fallis A. Data Structure and algorithms in java. Vol. 53, Journal of Chemical Information and Modeling. 2013. 1689-1699 p.

[5] Prim RC. Shortest Connection Networks And Some Generalizations. Bell Syst Tech J. 1957;36(6):1389–401.

[6] Grama A, Gupta A, Karypis G, Kumar V. Introduction to Parallel Computing, Second Edition. Communication. 2003. 856 p.

[7] P. Pacheco, An Introduction to Parallel Programming, Burlington, USA: Elsevier,2011.

[8] Shi Y, Lu J, Shi X, Zheng J, Li J. The Key Algorithms of Promoter Data Parallel Processing based on OpenMP. 2014;154–7.

[9] OpenMP Architecture Review Board, "OpenMP Application Program Interface", http://openmp.org/forum/

[10] Anjaneyulu GSGN, Dashora R, Vijayabarathi A, Rathore BS. Improving the performance of Approximation algorithm to solve Travelling Salesman Problem using Parallel Algorithm. 2014;337(3):334–7.

[11] Jasika N, Alispahic N, Elma A, Ilvana K, Elma L, Nosovic N. Dijkstra's shortest path algorithm serial and parallel execution performance analysis. MIPRO, 2012 Proc 35th Int Conv Inf Commun Technol Electron Microelectron. 2012;1811–5.

[12] Maroosi A, Muniyandi RC, Sundararajan E, Zin AM. Parallel and distributed computing models on a graphics processing unit to accelerate simulation of membrane systems. Simulation Modelling Practice and Theory. 2014;47:60-78. Available from, DOI: 10.1016/j.simpat.2014.05.005

[13] Ang MC, Aghamohammadi A, Ng KW, Sundararajan E, Mogharrebi M, Lim TL. Multi-core Frameworks Investigation on a real-time object Tracking application. J Theor Appl Inf Technol. 2014;70(1):163–71.

[14] Saabith ALS, Sundararajan E, Bakar AA. Parallel implementation of Apriori algorithms on the Hadoop-MapReduce platform - An evaluation of literature. Journal of Theoretical and Applied Information Technology. 2016 Mar 31;85(3):321-351.

[15] Maroosi A, Muniyandi RC, Sundararajan E, Zin AM. A parallel membrane inspired harmony search for optimization problems: A case study based on a flexible job shop scheduling problem. Applied Soft Computing Journal. 2016 Dec 1;49:120-136. Available from, DOI: 10.1016/j.asoc.2016.08.007

[16] Maroosi A, Muniyandi RC. Accelerated simulation of membrane computing to solve the N-queens problem on multi-core. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). PART 2 ed. Vol. 8298 LNCS. 2013. p. 257-267. (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Lecture Notes in Bioinformatics); PART 2). Available from, DOI: 10.1007/978-3-319-03756-1_23

[17] Setia R, Nedunchezhian A, Balachandran S. A new parallel algorithm for minimum spanning tree problem. Proc Int Conf High Perform Comput [Internet]. 2009;1–5. Available from: http://111.hipc.org/hipc2009/documents/HIPCSS09Papers/1569250351.pdf

[18] Loncar V, Skrbic S. Parallel implementation of minimum spanning tree algorithms using MPI. CINTI 2012 - 13th IEEE Int Symp Comput Intell Informatics, Proc. 2012;35–8.

[19] Dev C++. Available from: https://sourceforge.net/projects/orwelldevcpp/files/Setup Releases/Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup.exe/download

[20] UCI Machine Learning Repository: SGEMM GPU kernel performance Data Set [Internet]. [cited 2018 May 24]. Available from: https://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance