International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 7, July - 2013

Intelligent Transportation System By Parallelizing The Shortest Path Algorithm In Multi-Core Systems

M. S. Abirami, Dr. S. Andrews

[#1] *Asst. Professor, Department of Computer Applications, SRM University*
[#2] *Professor, Department of Information Technology, Mahendra Engineering College*

Abstract

In this paper, we examine the possible ways of quickly finding the shortest paths over real-road networks in an intelligent transportation system. This paper analyzes the performance of the shortest path program execution in serial and parallel way in multi-core systems. This research enables faster computation of optimal path planning in complex road networks. In this paper, we design a parallel Dijkstra's algorithm which is a challenging paradigm to test the efficiency in a multi-core architecture. Finally based on this study we conclude that the parallel programming is the most appropriate for multi-core systems which improves performance and simplicity of programming.

Keywords: *parallel shortest path algorithm, intelligent transportation, parallel computing, multi-core systems, performance measures.*

1. Introduction

The classic problem of finding single source shortest path (SSSP) over a network plays a significant role in many transportation related analyses [1]. In this paper we focus on satisfying the actual demands of quickly locating the most effective shortest paths over real-road networks in an intelligent transportation system [4]. We propose and implement a parallel shortest path algorithm. We also evaluate our parallel algorithm's performance using different multi-core systems. And from the experiments, it can be concluded that the parallel algorithm has good speed-up ratio and efficiency.

In this paper we face the challenges of parallelizing Dijkstra's algorithm for a multi-core architecture. The main objectives of this paper are, (i) to find the shortest path, (ii) to take the minimum execution time and (iii) to minimize the cost. The Dijkstra's algorithm maintains a set of visited nodes(S) for which the shortest path is already calculated. The set of unvisited nodes is implemented as a priority queue. It grows this set by selecting the unvisited node closest to s and distances of its neighbours are updated. This serializes a large part of the algorithm's operations, thus making Dijkstra's algorithm a hard to parallelize graph algorithm.

Path finding is an important concept for a variety of real world applications such as route guidance systems, robot navigation, traffic planning, optimal pipelining of VLSI chip, routing of telecommunication messages and decision-making. As the size of a graph increases, the computation time required to find the best traversal path increases exponentially. In a realistically modeled environment, this could mean extremely long wait times for shortest path finding. Parallel execution could split this job between multiple processors and help to overcome the inefficiencies arising from complex input data.

The implementation of these large scale problems on a single-core processor cannot satisfy the current computational requirements. For this reason, the high performance multi-core platform is used to deploy computationally intensive applications. To decompose a larger problem into several sub-problems and to map them to cores with the goal of the increasing performance is key fundamental factor of multi-core platform.

The rest of this paper is organized as follows. Section 2 presents the basics of SSSP and related work. Section 3 discusses the issues of SSSP. Section 4 details our parallel algorithm. In Section 5 we evaluate the algorithm's performance for large networks using multi-core platform. Finally, section 5 presents our conclusion and future work.

2. Background and Related Works

The Intelligent Transportation System (ITS) has been rapidly developed, and path optimization problems have been concerned by experts and

scholars [2]. LU Fen evaluates the serial shortest path algorithm; Ye Qing gets a pre-pruning algorithm based on Dijkstra algorithm. DEON, Pangcy classifies the algorithms according to solving shortest path problems. Zhan FB, Noon CE introduces the classic shortest path algorithms.

With the expansion of more complex solving large scale transportation networks, traditional serial computing has been unable to meet the demand of large-scale network to optimize the shortest path efficiently. Ji-Zhou Sun and Jin-Yan Chen give a method of predicting the maximum parallel execution time of n constituent tasks. Gwo-Jen Hwang et al propose to composite parallel test sheets efficiently to solve the practical application. A. K. Ziliaskopoulos et al have applied the shortest path algorithm in the label on PVM environment. But in the realization, multi-core multi-threaded parallel algorithms are not much implemented.

Multi-core architectures have been evolving as a means to increase processing power. They have quickly spread to all computing domains, from embedded systems to personal computers to high performance supercomputers. Multi-core architectures require parallel computation and explicit management of the memory hierarchy, both of which add programming complexity. So programmer or compiler explicitly parallelize the software is the key for enhance the performance on multi-core chip. A multi-core processor is a semiconductor chip on which multiple processor cores are implemented, with each processor core capable of executing an independent task. Parallel Programming is a form of computation in which program instructions are divided among multiple processors (cores, computers) in combination to solve a single problem, thus running a program in less time.

3. Issues and Solution using Graph Partitioning Method

In this section we present the problem definitions for Shortest Path Problems and we also discuss the general graph partitioning method [1] which is used for parallelizing the shortest path computations.

If Dijkstra's algorithm is used on a network containing an edge that has a negative value it does not work. We can use Dijkstra's algorithm to find the length of the shortest route between two vertices in a network. However, if we want to find the corresponding route, we need to record more information as we apply the algorithm. That is we find the route by backtracking through the network from the finishing point. Tracing a route through a network can be easily found if careful labeling of Dijkstra's algorithm is used.

The parallelized algorithm for shortest path computation mainly solves the problem by solving the same problem on smaller instances of the input. The graph partitioning is the major factor in determining the efficiency of the parallel algorithm and therefore it defines the number of necessary computation and communication operations. In other words, the more balanced the vertices are partitioned and the algorithm is parallelized, the better is the performance and efficiency of the resulting algorithm. In order to have a good efficiency each thread works concurrently on multi-cores.

4. Parallelizing Dijkstra's algorithm

Dijkstra's algorithm is a graph search algorithm that solves single-source shortest path for a graph with nonnegative weights. The single source shortest path (SSSP) problem is that of computing, for a given source vertex *s* and a destination vertex *t*, the weight of a path that obtains the minimum weight among all the possible paths [5].

For a weighted directed graph G=(V,E,w), |V| = n, |E| = m, and w be the non-negative value to the weight of each edge, the SSSP problem is to find the shortest path from a vertex v € to all other vertices in V. It maintains a set of nodes for which the shortest paths are known. It grows this set based on the node closest to source using one of the nodes in the current shortest path set. In order to obtain the routing table, we need O(V) rounds iterations (until all the vertices are included in the cluster). In each round, we will update the value for O(V) vertices and select the closest vertex, so the running time in each round is O(V). So, for the sequential Dijkstra's algorithm the total running time is $O(V^2)$. For the parallel Dijkstra's algorithm the total running time is $O(V^2/P + V. \log P)$, where P is the number of cores used. In order to obtain the routing table, we need O(V) rounds iteration (until all the vertices are included in the cluster). In each round,

we will update the value for O(V) vertices using P cores running independently, and use the parallel prefix to select the global closest vertex, so the running time in each round is O(V/P)+O(log(P)). So, the total running time is $O(V^2/P + V. \log P)$.

Pseudo-code for parallel Dijkstra's algorithm

```
V = set of all vertices in partition
S = empty set
for each v in V:
d[v] = infinity
previous[v] = undefined
d[s] = 0
Q = V
while Q is not an empty set:
q = extract-min(Q)
u = global-min(q)
if u is a member of set of V:
S = S union {u}
for each edge (u,v) incident with u:
if d[v] > d[u] + w[u,v]:
d[v] = d[u] + w[u,v]
previous[v] = u
```

In parallel formulation of Dijkstra's algorithm, the weighted adjacency matrix is partitioned. Each process selects, locally, the node closest to the source, followed by a global reduction to select next node.

Two parallel strategies are proposed. One to execute each of the n shortest path problems on a different processor (source partitioned), and the other to use a parallel formulation of the shortest path problem to increase concurrency (source parallel).

Source Partitioned Formulation uses n processors, each processor p_i finds the shortest paths from vertex v_i to all other vertices by executing Dijkstra's sequential single-source shortest paths algorithm. It requires no inter-process communication provided that the adjacency matrix is replicated at all processes. The parallel runtime of this formulation is $\theta(n^2)$. While the algorithm is cost optimal, it can only use n processors.

In Source Parallel Formulation each of the shortest path problems is further executed in parallel. We can therefore use up to n^2 processors. Given p processors (p > n), each single source shortest path problem is executed by p/n processors. Using previous results, this takes time: $T_p = \theta(n^3/p) + \theta(n \log p)$ that is the time for computation and communication. For cost optimality, we have used $p=O(n^2/\log n)$.

5. Experiment and Evaluation

5.1 Performance analysis

The parallel speedup and efficiency are defined as $S_p=T_s/T_p$ and $E_p= Sp/P \times 100\%$ respectively. T_s is the serial execution time, T_p is the parallel execution time and S_p is the speedup ratio and P is the number of processors.

5.2 Experimental setup

In this section, we discuss the performance evaluation of the sequential and parallel implementations of Dijkstra's algorithm with a network of large number of nodes using multi-core systems. Network with large data sets are taken for this experiment.

To evaluate our design, we use three hardware platforms. The first hardware platform consists of dual-core Opteron processor (4-cores) with 4GB main memory. The second platform is dual-core Intel Xeon processor (4-cores) which has 12GB of main memory. The third platform is Intel Core i7 processor has 8GB of main memory. Our parallel algorithm was implemented in Java programming language. This suggests that Parallel Dijkstra's algorithm outperforms on network graphs for any number of processors.

We analyse the sequential and parallel implementations with their execution time.

Number of Nodes	Quad Core	Dual Core	Single Core
100	0.0172	0.0266	0.0165
200	0.0350	0.0710	0.0632
300	0.0568	0.0980	0.1238
400	0.0691	0.1210	0.2630
500	0.0830	0.1560	0.4111

Table 1: Parallel Execution

No. of Nodes/ Cores	1	2	4	8	16	32
2500	0.075	0.053	0.044	0.051	0.084	0.241
10000	1.052	0.564	0.308	0.246	0.278	0.443
22500	5.241	2.692	1.4449	0.840	0.785	1.130

Table 2: Running time vs Number of cores

No. of Nodes/ Cores	1	2	4	8	16	32
2500	1	1.588	2.017	1.565	0.912	0.294
10000	1	1.883	3.403	4.405	3.886	2.372
22500	1	1.955	3.650	6.302	6.766	4.662

Table 3: Speed up vs Number of cores

No. of Nodes/ Cores	1	2	4	8	16	32
2500	0.074	0.091	0.141	0.342	1.173	7.380
10000	1.053	1.121	1.231	1.891	4.293	14.095
22500	5.241	5.390	5.766	6.651	12.411	36.055

Table 4: Cost vs Number of cores

The data in Table 1 and Table 2 represents the execution times taken by Dijkstra's sequential and parallel implementations for the networks of different sizes. The result shows that the parallel algorithm is efficient than their corresponding sequential algorithm. There is an improvement in parallel Dijkstra's code running in 2-cores, 4-cores etc., with large number of nodes.

We plot the graph using the data in Table 1 to analyse the performance of parallel algorithm which is presented in figure 1. Then we plot the graphs using the data in Table 2, Table 3 and Table 4 to analyse the performance measures such as execution time, speedup and cost with different multi-core architectures. The result obtained shows a vast difference in time required to execute the parallel algorithm and time taken by sequential algorithm. The parallel algorithm is approximately twice faster than the sequential.

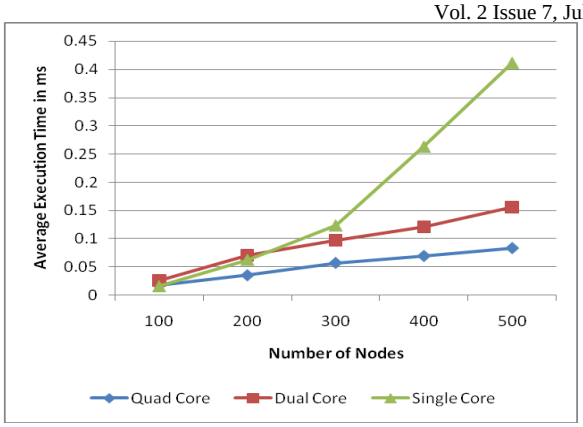

Figure 1: Parallel Execution

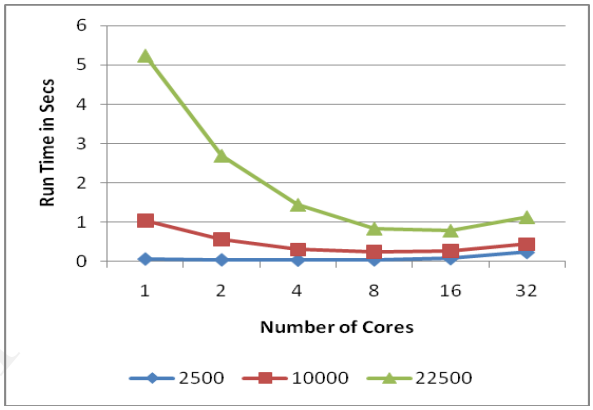

Figure 2: Running time vs Number of cores

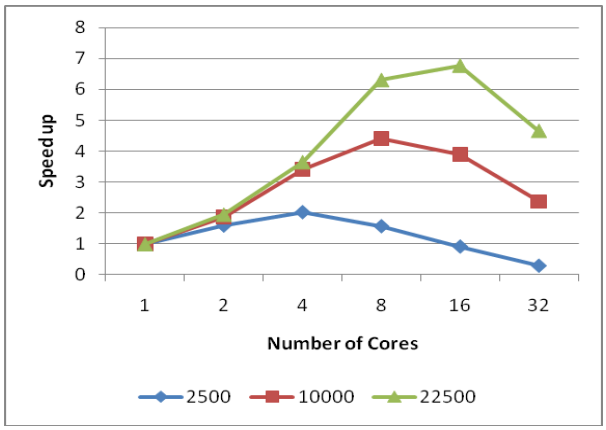

Figure 3: Speed up vs Number of cores

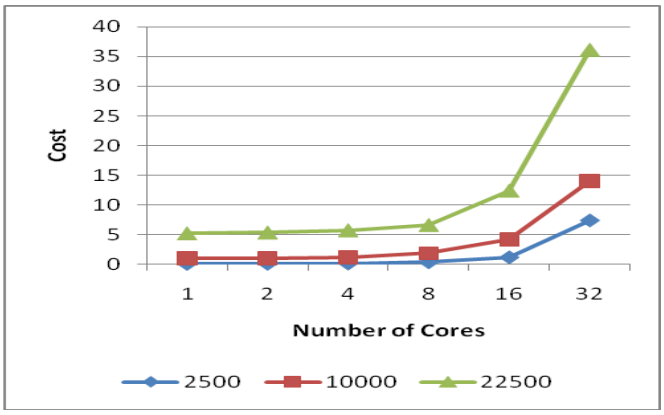

Figure 4: Cost vs Number of cores

5.3 Results

For 500 nodes, Average Speedup Ratio (Sp) = Average Serial Execution Time / Average Parallel Execution Time

Sp = 0.4111 / 0.156

Sp = 2.6353 (Dual Core)

Sp = 0.4111 / 0.083

Sp = 4.9530 (Quad Core)

Efficiency Ep = Speedup Ratio (Sp) / Number of Processors (p)

Ep= 2.6353 / 2

Ep = 1.3177 (Dual Core)

Ep = 4.9530 / 4

Ep = 1.2383 (Quad Core)

Therefore, average operating efficiency is 131.7% for Dual Core and 123.8% for Quad Core machines.

A closer look at the results reveals that for the large size network (22500), the running time is decreasing as the number of cores increases until it reaches the smallest value, then the running time will increase because of the communication latency.

For middle size network (10000), the phenomenon of a reducing running time is not that obvious. For a small size network (2500), the running time is even increasing as the number of cores increases, because the communication latency outperforms the benefit from using more cores. The speed up is increasing as the number of cores increases until it reaches the maximum value, then the speed up is decreasing. The speed up is increasing because of using more cores. The speed up is decreasing because the communication latency outperforms the benefit from using more cores. As the network size increases, the number of cores used to get the maximum speed up increases. (As shown in the figure, 2500-4 cores, 100000-8 cores, 22500-16 cores). The cost is increasing because the speed up (or the benefit of a reduced running time) cannot outperform the cost of using more cores.

6. Conclusion and Future Work

The results of the experiments lead us to several conclusions. First, this work proposes the parallel implementation of Dijkstra's algorithm. It is found that the execution time of parallel Dijkstra's algorithm increases as the number of nodes increases significantly and the execution time decreases as the number of processor increases. So, it is concluded that as the number of cores increased, parallel computing efficiency improved significantly. The parallel efficiency rapidly increased to above 90%. Second, the results reveals that for the large size network (22500 nodes), the running time decreases as the number of cores increases until it reaches the smallest value, then the running time will increase because of the communication latency. And the other performance measures speedup and cost are increasing until it reaches the maximum value, then they are decreasing because of using more cores.

We are going to run our test on a larger cluster real road-network environment and make a comparison with other parallel shortest path algorithms in the future. We also plan to implement our parallel algorithm in OpenMP parallel programming model by increasing number of threads to improve the performance on multi-core systems, and will be applied to the actual transportation network to achieve the expected results.

REFERENCES

[1] Y. Tang, Y. Zhang, H. Chen, "A Parallel Shortest Path Algorithm Based on Graph-Partitioning and Iterative Correcting", in Proc. of IEEE HPCC, pp. 155-161, 2008.

[2] Xin Fang, Han Cao, Zhao Lu, "Three ITS Path Algorithms OpenMP Parallel Optimization on Multi-core Systems", in Sixth International Conference on Fuzzy Systems and Knowledge Discovery, IEEE, 2009

[3] H. Chen, Y. Tang, and Y. Zhang, "Implementation and Evaluation of Graph-partitioning based Parallel Shortest Paths Algorithms", in *Journal of Computer Research and Development(Chinese)*, 2008.

[4] Chu-Hsing, Jui-Ling Yu, Jung-Chun Liu, Wei-Shen Lai, Chia-Han Ho, "Genetic Algorithm for Shortest Driving Time in Intelligent Transportation

Systems", in International Journal of Hybrid Information Technology, Vol. 2, No. 1,2009.
[5] Guruprasad Nagraj, Y.S. Kumaraswamy, "Serial and Parallel Implementation of Shortest Path Algorithm in the Optimization of Public Transport Travel", in International Journal of Computer Science Engineering and Information Technology Research, Vol. 1, 72-87, 2011.
[6]] K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, "Parallel Shortest Path Algorithms for Solving Large-Scale Instances", *9th DIMACS Implementation Challenge -- The Shortest Path Problem*, DIMACS Center, Rutgers University, Piscataway, NJ, Nov. 2006.
[7] L. Fu, D. Sun, L.R. Rilett, "Heuristic Shortest Path Algorithms for Transportation Applications: State of the Art", in Computers and Operations Research 33, Elsevier, 3324-3343, 2006
[8] N. Edmonds, A. Breuer, D. Gregor, and A. Lumsdaine. Singlesource shortest paths with the parallel boost graph library. In *9thDIMACS Implementation Challenge*, 2006.
[9] R. Kalpana, P.Thambidurai, "Combining Speedup Techniques based on Landmarks and Containers with Parallelised Pre-processing in Random and Planar Graphs, in International Journal of Computer Science and Information Technology, Vol. 3, No. 1, 2011.
[10] Yongtaek LIM, Hyunmyung KIM, "A Shortest Path Algorithm for Real Road Network based on Path Overlap", in Journal of the Eastern Asia Society for Transportation Studies, Vol 6, 1426-1438, 2005.
[11] U. Meyer, "Design and Analysis of Sequential and Parallel Single-Source Shortest-Paths Algorithms", PhD thesis, Universit¨at Saarlandes, Saarbr¨ucken, Germany,October 2002
[12] Goldberg, A. 2001b. A simple shortest path algorithm with linear average time. In *9th Ann. European Symp. on Algorithms(ESA 2001)*. Lecture Notes in Computer Science, vol. 2161. Springer, Aachen, Germany, 230–241.
[13] Hribar, M. R., Taylor, V. E. and D. E. Boyce, Implementing parallel shortest path for parallel transportation applications, *Parallel Computing*, Vol. 27, 1537-1568, 2001.
[14] I.Watson, C. Kirkham, andM. Lujan. A study of a transactional parallel routing algorithm. In *PACT*, 2007.
[15] B. Hendrickson, J.W. Berry, "Graph Analysis with High-Performance Computing", *Computing in Science and Engineering*, vol.10, no. 2, pp.14-19, Mar/Apr, 2008.
[16] Goldberg . A, "Shortest path algorithms: Engineering aspects", in *Proc. of 12th International Symposium on Algorithms and Computation*, Springer-Verlag, London, UK, 502–513, 2001.
[17] A. Crauser, K. Mehlhorn, U. Meyer, P. Sanders, "A parallelization of Dijkstra's shortest path algorithm", in Proc. of MFCS, pp. 722-731, 1998.
[18] G.S. Brodal, J. L. Traff, C.D. Zaroliagis, I. Stadtwald, "A Parallel Priority Queue with Constant Time Operations", in *Journal of Parallel and Distributed Computing*, 49:4–21, 1998.
[19] F.B. Zhan, "Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures", in *Journal of Geographic Information and Decision Analysis* 1(1): 69-82, 1998.
[20] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. "A parallelization of Dijkstra's shortest path algorithm", in Mathematical Foundations of Computer Science (MFCS), Vol. 1450, Pg. 722–731, Springer, 1998.
[21] U. Meyer and P. Sanders, "Delta-stepping: A parallel single source shortest path algorithm", in Proceedings of the 6th Annual European Symposium on Algorithms, pages 393–404, Springer-Verlag, 1998.
[22] Brodal G, Tr¨aff J, Zaroliagis C, "A parallel priority queue with constant time operations", in *Journal of Parallel and Distributed Computing 49, Vol.* 1, 4–21, 1998.
[23] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. Math. Program., 73:129–174, 1996.
[24] G.C. Hunt, M.M. Michael, S. Parthasarathy, and M.L. Scott. An efficient algorithm for concurrent priority queue heaps. *Inf.Proc. Letters*, 60:151–157, 1996.