Comparative Analysis of Parallelised Shortest Path Algorithms using Open MP

Uditi
Electronics and Communication Engineering
VIT University
Vellore, Tamil Nadu, India
uditi12@yahoo.co.in

Arun M
Embedded Systems, Associate Professor
VIT University
Vellore, Tamil Nadu, India

***Abstract*— this paper presents parallel implementations and includes performance analysis of three prominent graph algorithms (i.e., Bellman Ford, Floyd-Warshall and Dijkstra) used for finding the all-pairs of shortest paths. The algorithm implementations were parallelized using Open MP (Open Multi-Processing). Their performances were measured on 4 different configurations i.e. dual core i3, quad core i5, quad core i7 and 8 core processors. This paper also presents a comparative study of serial and parallel implementations of these algorithms keeping execution time and number of graph nodes as the parameter. Finally, the results show that, execution time can be reduced using parallel implementation for larger number of graph nodes. Also, the conclusions are drawn for the best algorithm to be used which works for all the graph nodes with less execution time.**

Keywords—Shortest path algorithms; Open MP; Parallel computing; execution time

I. Introduction

Shortest-paths problems are the most fundamental and also the most commonly encountered graph problems, both in themselves and as sub problems in more complex settings. With the development of technology, the research about graph theory gets a wide range of attention, and a variety of graph structures and algorithms have been proposed. The shortest path algorithm is always a research hotspot in graph theory. Shortest path algorithm is used to implement traffic engineering in IP networks and to improve Intelligent and Transportation Systems. They are applied to many fields, such as operations research, plant and facility layout, robotics, transportation, logistics, VLSI design and so on. These are frequently needed in telecommunications and transportation industries, where messages or vehicles must be sent between two geographical locations as quickly or as cheaply as possible. Other examples are complex traffic flow simulations and planning tools, which rely on a large number of individual shortest paths problems. There are many shortest path algorithms for solving the graph problems. The more famous of them are Dijkstra, Bellman Ford and Floyd Warshall algorithm. With the development of semiconductor technologies and the advent of the multi-core processor, Parallel Computing technology is emerging and becoming the mainstream now. In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized. The computational complexities (i.e., computing power and memory requirements) of the Dijkstra, Bellman Ford and Floyd-Warshall algorithms grow with an increasing number of nodes and edges for a given graph network. In general, there are two computing approaches for solving such a problem i.e. serial computing and parallel computing. In serial computing, a given problem is broken into discrete parts and discrete parts are solved sequentially (i.e., one at a time). In contrast, parallel computing attempts to solve the discrete parts of a problem concurrently. Currently the algorithms for the serial shortest path optimization have reached the time limitation. Therefore the parallel computation is an efficient way to improve the performance. There are various parallel programming models and parallel programming environments, such as Message Passing Interface (MPI), Open Multi-Processing (Open MP), Parallel Virtual Machine (PVM) and so on. In this thesis, Open Multi-Processing (Open MP) is used to implement the above mentioned algorithms and analyze their performance. These systems are scalable, i.e., they can be tuned to available budget and computational needs and allowed efficient execution of both demanding sequential and parallel

978-1-5090-4929-5©2017 IEEE

applications. By putting some constraints on the data and taking the advantage of the hardware, the performance of the algorithms can be significantly improved. The parallel efficiency can be defined as a ratio of the run time of a parallel algorithm to the run time of a serial algorithm.

II. Algorithms

There are various Shortest Path Algorithms and out of which three algorithms i.e. Dijkstra algorithm, Floyd Warshall algorithm and Bellman Ford Algorithm are prominent and have been researched upon. In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized.

I. FLOYD-WARSHALL ALGORITHM

In computer science, the **Floyd–Warshall algorithm** is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices. Although it does not return details of the paths themselves. The algorithm works as follows: [12]

- This algorithm find the shortest path by constructing the distance matrix from the source node to all the other nodes. The distance matrix follow the following things
  - **i.** If i=j, distance should be 0
  - **ii.** Distance[i][j] should be equal to Distance[j][i].
  - **iii.** If two nodes are not connected, distance should be infinity i.e. a very large number.
- Then the weighted matrix is generated according to the formula
  $W_{ij}^{k} = \min (W_{ij}^{k}, W_{ik}^{k-1} + W_{kj}^{k-1})$
- Now, construct a matrix D1, in which first row and column will remain the same and the other values will be filled by the above mentioned formula.
- Now, construct matrix D2, in which second row and second column of D1 will remain the same and other values will be filled by the above formula.
- Repeat this for all the rows and columns i.e. repeat this n times where n is equal to the number of edges.
- The final matrix will be shortest distance matrix.

II. DIJKSTRA'S ALGORITHM

Dijkstra's algorithm finds the length of an shortest path between two vertices in a graph. The algorithm works as follows: [10]

- It maintains a list of unvisited vertices
- It chooses a vertex (the source) and assigns a maximum possible cost (i.e. infinity) to every other vertex
- The cost of the source remains zero as it actually takes nothing to reach from the source vertex to itself
- In every subsequent step of the algorithm it tries to improve (minimize) the cost for each vertex. Here the cost can be distance, money or time taken to reach that vertex from the source vertex. The minimization of cost is a multi-step process.
- For each unvisited neighbor of the current vertex calculate the new cost from the vertex.
- For e.g. the new cost of vertex 2 is calculated as the minimum of the two or (sum of cost of vertex 1 + the cost of edge from vertex 1 to vertex 2)
- Select a vertex from the list of unvisited nodes (having smallest cost) and repeat step 4.
- At the end there will be no possibilities to improve it further and then the algorithm ends

III. BELLMAN FORD ALGIRHTM

The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. The algorithm works as follows: [11]

- From the given source vertex, initialize all distances as infinite, except the distance to source itself. The number of times the edges must be processed is equal to the total number of vertices in the graph.
- Starting from the source node, calculate the distance to the other nodes which are directly connected to the source node. In other words, visit the nodes which are one edge long to the source node and calculate their distance.
- The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time
- The second iteration guarantees to give all shortest paths which are at most 2 edges long.
- Repeat the algorithm for n number of times where n is the total number of edges in the graph and the final result would be the shortest distance from the source node to all the other nodes.

III. Parallelizing Algorithms

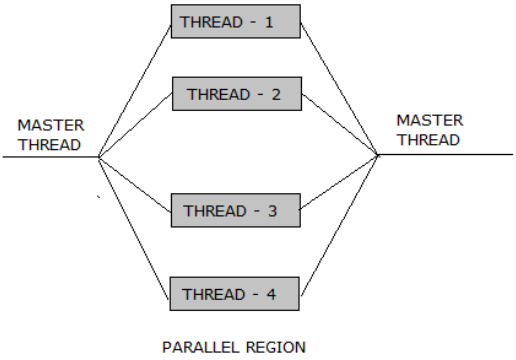
I. PARALLEL DIJKSTRA'S ALGORITHM

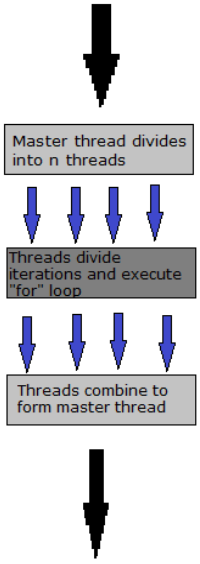
II. PARALLEL BELLMAN FORD ALGORITHM

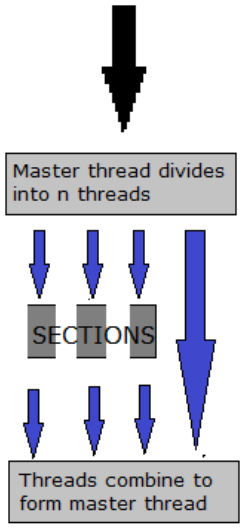
III FLOYD WARSHALL ALGORITHM

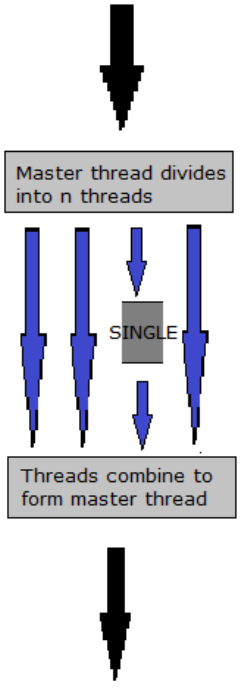
978-1-5090-4929-5©2017 IEEE

All the above three algorithms use a parallel formulation of the shortest path problem to increase concurrency. These algorithms are parallelized using Open MP. An Open MP application begins with a single thread called the master thread. As the parallel program executes, the master thread encounters parallel regions in which the master thread creates thread teams (which include the master thread). The number of threads which are formed is equal to the number of cores the system has on which the code is being implemented. At the end of a parallel region, the thread teams are stopped and all the thread combine to form the master thread and continues execution. Since Open MP is primarily a pragma or directive based standard. To start, serial code of the algorithms were written and then were parallelized keeping in mind the regions which are independent and can be parallelized. For parallelization using open MP, several constructs like work sharing constructs (for directive, section directive and single directive), synchronization constructs (critical directive and barrier directive) and clauses like no wait, private, shared are used. Also measure the computation time Open MP has a function OMP_GET_WTIME() is used.

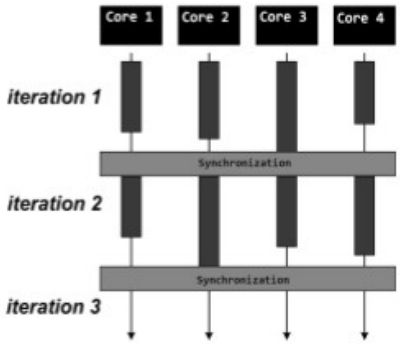


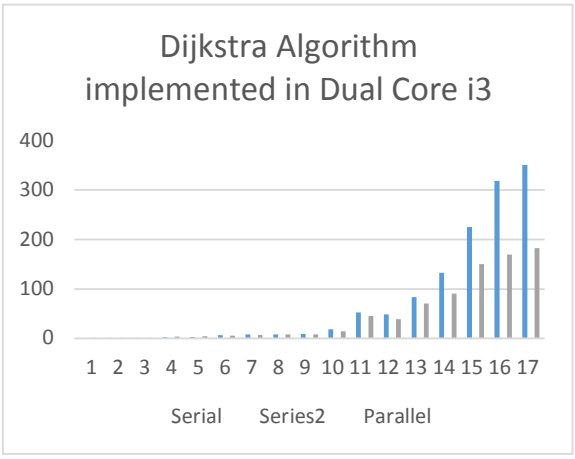






978-1-5090-4929-5©2017 IEEE





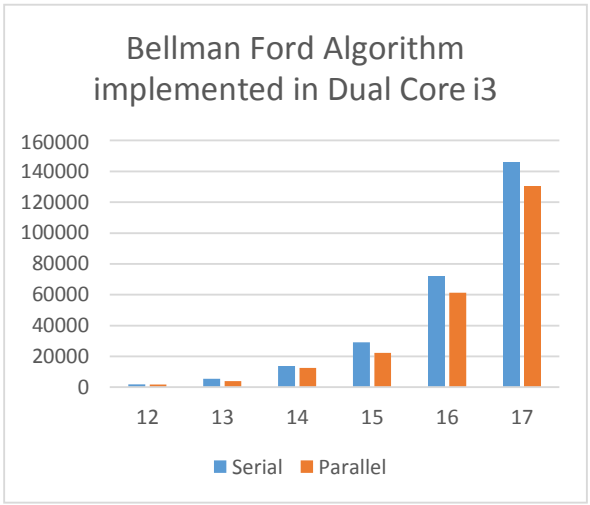
IV. Experimental analysis and Results

The algorithms were evaluated taking their computation time as a parameter. They were tested on systems with four different configurations which are as follows:

a. Intel® Dual Core i3 @1.9GHz
b. Intel® Quad Core i5 @2.6GHz
c. Intel® Quad Core i7 @2.4GHz
d. Intel® 16-core system

All the programs are separately tested on all these configurations and then their performance is evaluated.

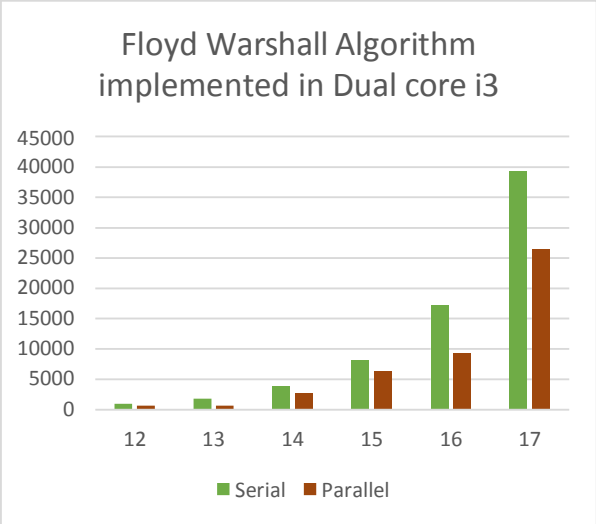
Table 1: *Implementation done in* Intel® Dual Core i3 @1.9 GHz



978-1-5090-4929-5©2017 IEEE

S. No	No. of graph nodes	Execution time (s)					
		Dijkstra (serial)	Dijkstra (parallel)	Bellman Ford (serial)	Bellman Ford (parallel)	Floyd Warshall (serial)	Floyd Warshall(parallel)
1	10	0.00112	0.002579	0.000244	0.0068	0.0066	0.007
2	50	0.00381	0.004181	0.0134	0.0183	0.003	0.018
3	100	0.0285	0.0268	0.0229	0.0297	0.0175	0.05
4	500	0.172	0.251	1.564	1.583	1.966	1.923
5	600	0.633	0.687	2.368	3.152	2.013	2115
6	700	0.695	0.852	3.689	4.763	3.138	3.403
7	800	0.81	0.919	4.905	6.545	5.26	5.409
8	1000	0.929	0.903	10.855	10.696	23.51	23.25
9	2000	3.76	2.958	93.392	90.785	65.495	64.032
10	3000	11.431	8.833	270.372	259.179	226.893	215.826
11	4000	23.612	17.94	768.138	754.125	511.07	451.237
12	5000	35.68	20.487	1985.431	1898.604	990.99	709.583
13	7000	76.048	52.249	5371.892	5119.437	1845.294	9690.312
14	9000	124.073	109.475	13813.532	12567.53	3901.263	2702.129
15	11000	186.166	165.966	29153.981	22179.78	8130.927	6277.424
16	13000	362.491	229.07	71923.419	61159.58	17259.36	9283.12
17	15000	482.73	310.89	145839.83	129892.4	39129.48	26429.42

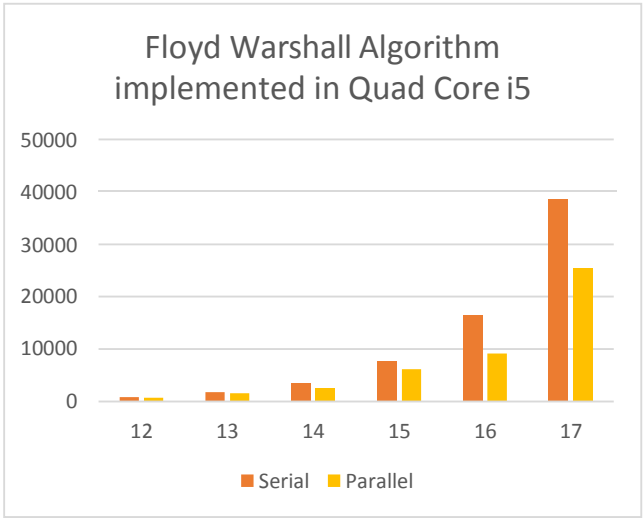
978-1-5090-4929-5©2017 IEEE



978-1-5090-4929-5©2017 IEEE

Table 2: *Implementation done in* Intel® Quad Core i5 @2.6 GHz

S. No.	Number of graph nodes	Execution time (s)					
		Dijkstra (serial)	Dijkstra (parallel)	Bellman Ford (serial)	Bellman Ford (parallel)	Floyd Warshall (serial)	Floyd Warshall (parallel)
1	10	0.0007	0.0227	0.0024	0.0083	0.0056	0.0084
2	50	0.0041	0.00585	0.0048	0.0039	0.0029	0.0168
3	100	0.0908	0.0917	0.0209	0.0244	0.0141	0.049
4	500	3.037	3.222	1.266	1.247	1.891	1.916
5	600	4.411	4.606	1.706	1.745	1.993	2.029
6	700	4.794	6.289	2.573	2.493	2.991	3.253
7	800	5.171	8.152	3.895	3.806	5.129	5.217
8	1000	12.763	9.689	8.154	7.859	21.721	20.182
9	2000	11.916	-15.621	76.437	76.081	57.128	48.018
10	3000	15.727	15.621	273.08	259.62	176.139	153.148
11	4000	26.301	26.603	671.36	647.665	471.294	439.149
12	5000	40.789	31.672	1584.7	1119.01	816.284	724.529
13	7000	86.83	61.684	4309.6	3812.76	1724.294	1539.19
14	9000	140.55	107.218	13129	12490.1	3438.193	2622.11
15	11000	212.96	162.019	27823	21998.1	7626.194	6149.29
16	13000	343.48	224.619	67926	61312.7	13499.194	9133.18
17	15000	411.05	299.739	140761	128193.	38721.193	25329.2



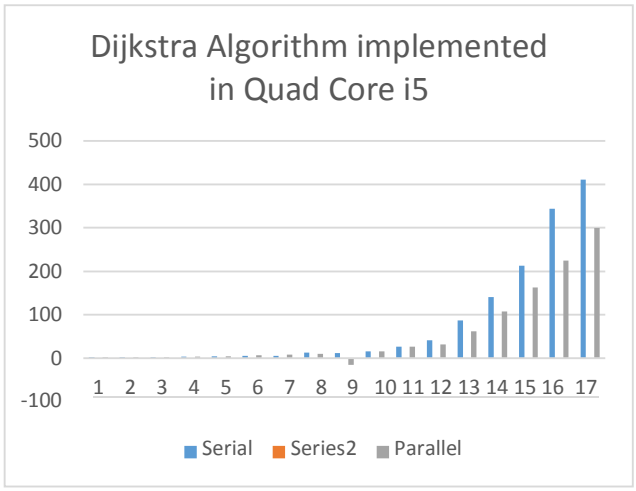
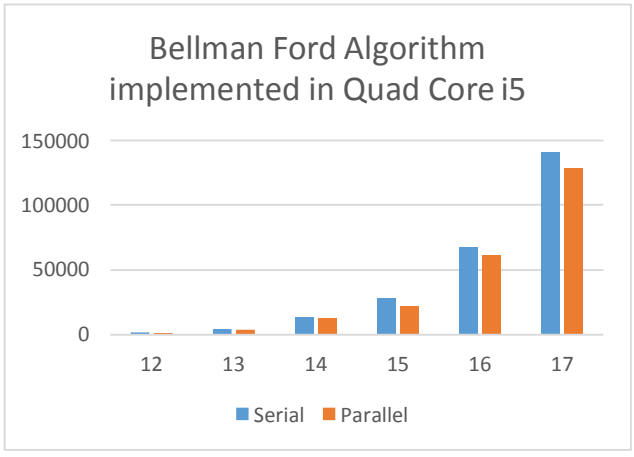
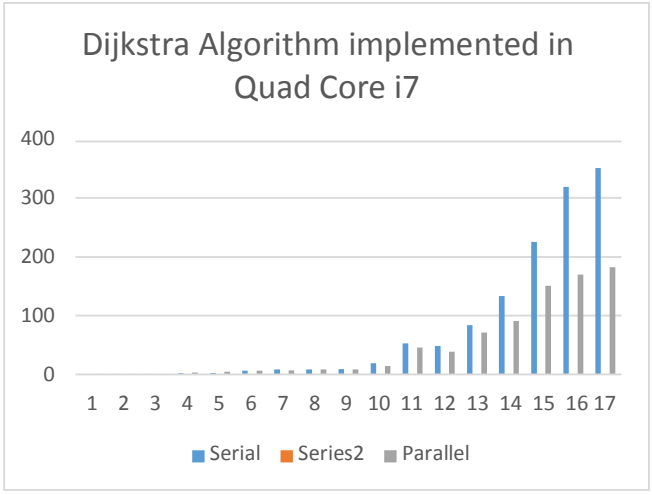


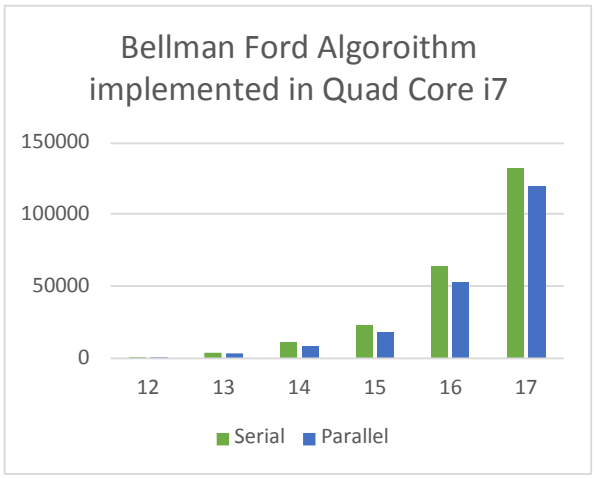
Table 3: *Implementation done in* Intel® Quad Core i7 @2.4 GHz

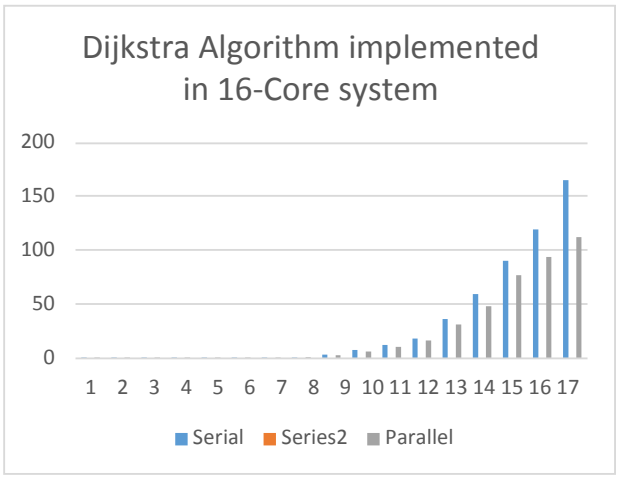
S.No	Number of graph nodes	Execution time (s)					
		Dijkstra(serial)	Dijkstra (parallel)	Bellman Ford (serial)	Bellman Ford (parallel)	Floyd Warshall (serial)	Floyd Warshall (parallel)
1	10	0.0001	0.00872	0.002231	0.0023	0.0051	0.008
2	50	0.0011	0.00213	0.003461	0.0023	0.0023	0.0128
3	100	0.0606	0.0928	0.0169	0.0215	0.01275	0.0315
4	500	2.298	3.232	0.913	1.672	1.766	1.835
5	600	2.82	4.59	1.291	1.457	1.904	2.115
6	700	6.265	6.256	2.012	2.278	2.819	3.145
7	800	8.149	7.407	3.013	3.512	4.831	5.116
8	1000	8.159	8.155	7.821	6.157	18.691	17.95
9	2000	9.166	8.249	67.931	52.671	41.829	39.641
10	3000	18.92	14.65	157.931	134.68	145.492	123.925
11	4000	52.574	45.69	534.126	492.152	440.217	391.217
12	5000	48.724	38.901	981.725	719.014	789.491	691.462
13	7000	83.722	71.136	3732.174	3356.137	1572.15	1252.419
14	9000	132.64	90.516	11322.13	8125.32	2983.19	2526.196
15	11000	225.12	150.238	23126.57	17924.46	6911.82	6184.196
16	13000	318.30	161.421	63956.11	53192.71	14485.4	9843.37
17	15000	350.23	182.123	131665.5	119106.5	35249.3	22468.21

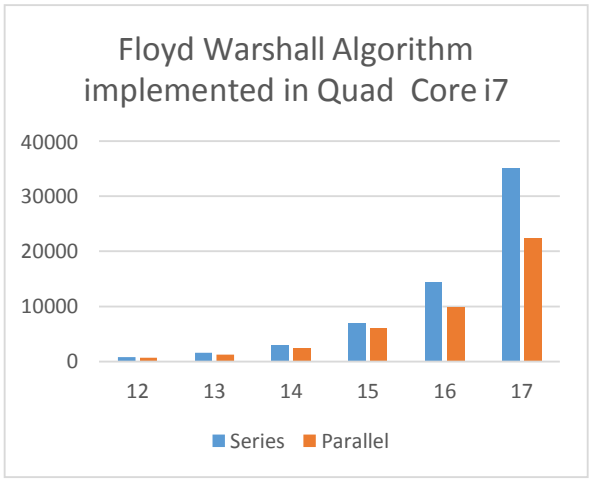




978-1-5090-4929-5©2017 IEEE







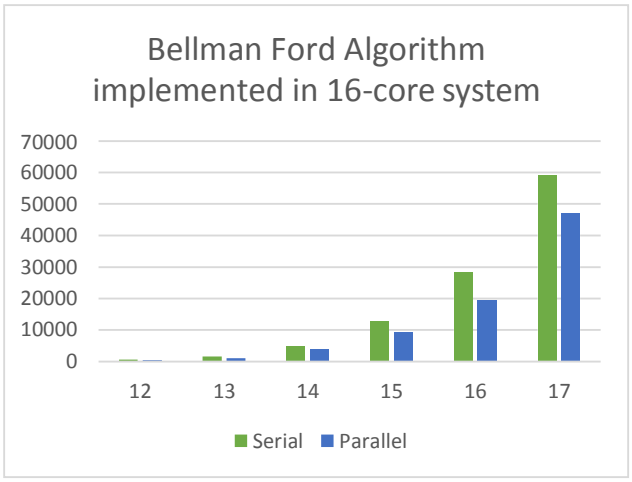
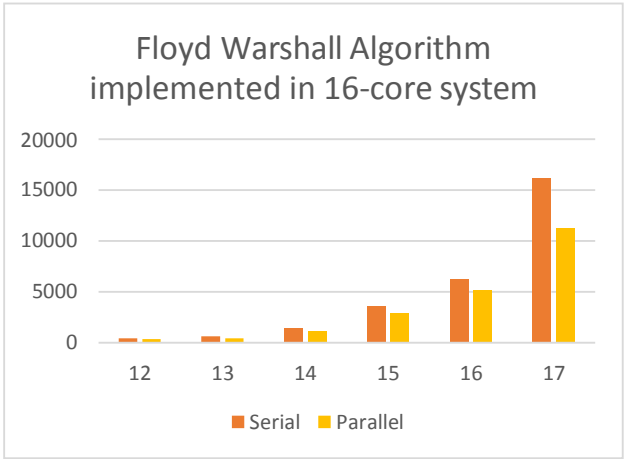


***Table 4**: Implementation done in 16-core system*

S.No	Number of graph nodes	Executio n Time					
		Dijkstra (serial)	Dijkstra (parallel)	Bellman Ford (serial)	Bellman Ford (parallel)	Floyd Warshall (serial)	Floyd Warshall (parallel)
1	10	0.0001	0.091	0.0002	0.042	0.00023	0.00049
2	50	0.0008	0.079	0.001709	0.058	0.00124	0.00185
3	100	0.0085	0.099	0.018	0.067	0.00632	0.00715
4	500	0.159	0.279	0.776	0.977	0.193	0.235
5	600	0.207	0.373	1.21	1.633	0.473	0.755
6	700	0.358	0.455	1.7	2.405	1.027	1.045
7	800	0.4233	0.624	2.559	3.091	2.384	2.394
8	1000	0.7406	0.871	4.456	3.721	8.184	7.835
9	2000	3.401	2.723	48.507	47.9	28.184	25.143
10	3000	7.754	6.295	155.469	142.49	103.146	85.284
11	4000	12.255	10.752	367.772	325.48	238.184	391.217
12	5000	18.224	16.36	618.479	517.47	419.495	323.184
13	7000	36.366	31.267	1594.842	987.15	629.294	423.853
14	9000	59.416	48.128	4729.302	3958.7	1395.18	1135.184
15	11000	89.911	77.033	12945.59	9284.3	3619.19	2943.184
16	13000	118.96	93.648	28304.94	19473.	6293.39	5183.932
17	15000	164.47	111.784	59284.48	47294	16228.1	11242.184



978-1-5090-4929-5©2017 IEEE

Based on the experimental analysis, it was concluded that Dijkstra Algorithm is the best algorithm as complexity of Dijkstra algorithm (O(n)) is way less than the complexity of Bellman Ford ($O(n^3)$) and Floyd Warshall algorithm ($O(n^3)$). Also, parallel execution works better than serial execution in terms of computation time if the number of graph nodes is greater than 1000. For number of nodes lesser than 1000, serial execution works faster.

16-core system has the lowest computation time. More the system's clock speed, less time it takes to compute the algorithm. Thus computation time in 16 core < computation time in Quad core i7 < computation time in Quad core i5 < computation time in Dual Core i3.

V. Conclusions

Parallel programming is a very intricate, yet increasingly important task as we have entered the multicore era and more cores are made available to the programmer going from dual core to 32 and 64 core systems or super computers. Independent tasks within a single application can be easily mapped on multicore platforms, the same is not true for applications that do not expose parallelism in a straightforward way. According to the results achieved, Bellman Ford algorithm is a challenging example of such an algorithm that is difficult to accelerate when executed in a multithreaded fashion because of its complexity being $O(n^3)$. It is very important to have in mind the two major issues inherent to Bellman Ford algorithm: limited independent tasks and excessive synchronization.

There are several possible reasons why the parallel execution of algorithms wasn't as fast as expected for smaller number of nodes. One of the reasons could be that the code used may be inefficient. Another possibility is that, for a small amount of nodes, more time could be spent on parallelization and synchronization then it is spent on execution of code as sequential.

There are several areas in which this research could be extended. Overcoming issues such as utilizing shared memory, avoiding branches, choosing appropriate data structures and designing algorithms correctly are the key to success with parallelization of these three algorithms.

VI. Acknowledgment

During this project, there were several individuals and organizations that encouraged and gave their support to this work. I want to express our gratitude, especially

— To my supervisor and guide Prof Arun M, for the opportunity he gave me to make this work possible. I am particularly thankful for the excellent support and supervising work as a result of his long experience.
— To friends who directly or indirectly supported me because I could not mention all of them, we want to express here our grateful acknowledgements.
— Finally, I wish to thank our families for all the patience and support given during the time spent during the project.

.

VII. References

[1] Jasika, Nadira, et al. "Dijkstra's shortest path algorithm serial and parallel execution performance analysis." *MIPRO, 2012 proceedings of the 35th international convention*. IEEE, 2012.

[2] Yin, Chao, and Hongxia Wang. "Developed Dijkstra shortest path search algorithm and simulation." *Computer Design and Applications (ICCDA), 2010 International Conference on*. Vol. 1. IEEE, 2010

[3] Lee, Kyung Min, et al. "OpenMP parallel programming using dual-core embedded system." *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*. IEEE, 2011.

[4] Cilardo, Alessandro, et al. "Efficient and scalable OpenMP-based system-level design." *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE, 2013.

[5] Pradhan, Anu, and G. Mahinthakumar. "Finding all-pairs shortest path for a large-scale transportation network using parallel Floyd-Warshall and parallel Dijkstra algorithms." *Journal of Computing in Civil Engineering* 27.3 (2012)

[6] Jian, Ma, Li Ke-ping, and Li-yan Zhang. "A parallel floyd-warshall algorithm based on tbb." *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*. IEEE, 2010.

[7] Chakaravarthy, Venkatesan, et al. "Scalable single source shortest path algorithms for massively parallel systems." *IEEE Transactions on Parallel and Distributed Systems* (2016).

[8] Kulkarni, Pranav, and Sumith Pathare. "Performance analysis of parallel algorithm over sequential using OpenMP." *IOSR Journal of Computer Engineering (IOSR-JCE)* 16.2 (2014):58-62.

[9] Mohammad, Asad, and Vikram Garg. "Comparative Analysis of Floyd Warshall and Dijkstras Algorithm using Opencl." *International Journal of Computer Applications* 128.17 (2015).

[10] Djikstra Algorithm (online). Available : https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[11] Bellman Ford Algorithm (online). Available : https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

[12] Floyd Warshall Algorithm (online). Available: https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

978-1-5090-4929-5©2017 IEEE