International Journal of Computer Applications (0975 – 8887)
Volume 47– No.24, June 2012

Performance Analysis of Parallel Speedup Techniques for Shortest Path Queries in Networks of Random and Planar Types

R.Kalpana
Department of CSE
Pondicherry Engineering College
Puducherry

P.Thambidurai
Phd,Perunthalaivar Kamarajar Institute of Engineering & Technology
Karaikal
Puducherry

ABSTRACT

The essential elements of any network application system uses shortest-path algorithm mostly for problems of network namely routing, viz. When seen in the light of the basic requirement of such a system, to provide high quality path identification or routing solutions fast, algorithms have to be efficient. There are many speedup techniques and combined speedup techniques available which find shortest path efficiently in networks. Also parallelization is incorporated in some of the speedup techniques, where the performance is monitored in multicore processors. This paper deals with comparison of parallelized speedup techniques with sequential version of the same and finding performance improvement achieved in parallelized speedup techniques with respect to runtime and number of vertices visited during shortest path computation. The techniques were tested in random and planar types of graph networks, which may be suitable for networks of the same type. Performance of parallelization has good impact of speedup in random graph type of networks(45% to 90% with respect to runtime and 25% to 830% with respect to vertices visited) than planar graph type of networks.

General Terms

Algorithms, Parallel Programming

Keywords

Dijkstra's algorithm, Shortest path computation, Speedup Techniques, Parallel Speedup.

1. INTRODUCTION

In the context of routing in scheduled vehicles like train, buses, web search engines and navigation applications the computation time is very important component. One of the essential component of these applications is the shortest path computation time. In these applications, different places are considered as nodes and their distances are considered as edge weight, which constitute a graph structure. The shortest path queries for such applications were originally solved by Dijkstra[1], Bellman-ford, and Johnson. Dijkstra's algorithm implemented with Fibonacci heaps is still the fastest known algorithm for the general case of arbitrary nonnegative edge lengths, taking $O(m + n \log n)$ worst-case time. There are several Speed up techniques available to improve the speed up of shortest path computation with Dijkstra's algorithm[1],[2]. The basic speeds up techniques are bidirectional search[3], goal directed search[4],[5], multilevel approach[6],[7],[8],[9],[10], shortest-path container[11], arc flags[12] and reach based method[13]. The basic speedup techniques can also be combined in different flavors and their performance were also improved. These basic speedup techniques[14] and combined speedup techniques[15],[16] cannot be always guaranteed to prove to be faster than the original Dijkstra's algorithm. However it can be empirically shown that they certainly improve the speedup of many of the applications[2]. The shortest path problem has two phases of implementation for applications where there is a need for voluminous data sets. They are pre-processing phase and shortest path computation phase. Pre-processing techniques were identified to make the applications to work fast. It makes to work fast in very large networks, where there is a need for many 1 to n shortest path computations. The speed up factor is found to be high in techniques where pre-processing the network is done at the design phase of the network itself[2],[15]. In the shortest path computation phase, actual speedup techniques integrated with Dijkstra's algorithm works to give the result in optimal time. When these speedup techniques were parallelized it shows an improvement in preprocessing time, runtime and number of vertices visited for some of the techniques.

All programs contain some regions that are suitable for parallelization and other regions that are not. By using an increasing number of processors, the time spent in the parallelized parts of the program is reduced, but the sequential section remains the same. Eventually the execution time is completely dominated by the time taken to compute the sequential portion, which puts an upper limit on the expected speedup. This effect, known as *Amdahl's law[17]*, can be formulated as

$$S = \frac{1}{(f_{par}/P + (1 - f_{par}))} \qquad (1)$$

Where f_{par} is the parallel fraction of the code and P is the number of processors. In the ideal case when all of the code runs in parallel, f_{par} = 1, the expected speedup is equal to the number of processors.

A new parallel approach called parallel profile search algorithm for best connections between stations in public transportation networks was designed and implemented in multicore servers[18]. A good speedup in computation time and settled nodes is achieved in public transportation networks. Parallelism in sequential Dijkstra implementation of general networks is exploited at the level of parallel edge relaxations and parallel priority queuing in PRAM models and Communication networks[19],[20],[21],[22]. Results of shared-memory parallel variants of the multi-level overlay graph construction necessary for HNR are discussed in [23] and overlay graphs were experimented in eight core machines. A high number of updates per time are desirable to keep the replies to the shortest path queries as up-to-date as possible. On multicore processors, the repeated precomputation step for HNR takes roughly two minutes. The results of parallelized bidirectional search [24], parallelized landmarks[25], parallelized bidirectional arc flags[26], and parallelized multilevel arc flags[27] on multicore processors are equivalently very good which aims for comparison of the results of the speedup techniques

In this paper, the parallelized speedup techniques were analysed individually, and experimented in random and planar graphs. A comparative study was made for all the five speedup techniques by considering sequential and parallel version of the same technique with respect to the output parameters runtime and vertices visited during shortest path computation. The remainder of the paper is organized as follows. Section 2 describes the parallelized speedup techniques and the analysis of the same. . Section 3 describes empirical evaluation of the sequential and parallel speedup techniques using random and planar graph types using LEDA library. Finally, Section 4 draws conclusions and gives directions for future work

2. DESIGN OF PARALLELIZATION IN SPEEDUP TECHNIQUES

The parallel version of the Dijkstra's algorithm with speedup techniques were parallelized in its initialization phase and update phase. This parallelization is achieved using OpenMP constructs in the iterative statements (for loops). Depending on the number of processors run in the system, the iterative operations will be executed in parallel using multithreaded approach. Each process P_i (1<=i<=k) (k being the number of processors in the system) initializes the priority queue Q partly. For large value of n, the operations of initialization phase and update phase were shared by multiple processors. This takes a time t which is less than the time achieved in sequential speedup techniques. If the node size is very less, then the time for communication between threads is eaten by the computation time. Even then the worst case running time of parallelized speedup techniques will be less than the time achieved in sequential speedup techniques, which is less than O(n+m log n). The various phases of each speedup technique and different phase of the same which can be parallelized is given in Table I.

2.1 Parallel Bidirectional Search

The sequential version of Bidirectional search, which terminates when a node is marked permanent by both searches i.e $u \in$ perm or $ru \in$ perm. The search is processed using two priority queues Q and rQ. Bidirectional search involves initialization phase, node selection phase and update phase. Initialising the priority queue Q and reverse priority queue rQ takes a running time of O(n) each for n nodes. In node selection phase, it takes approximately n/2 comparisons for selecting a minimum distance node from the priority queue. The node selection will be ended when the perm node is marked by any of the other search. In update phase it takes O(n) for each variant of Bidirectional search. In both searches the search tree expands with a branching factor b and the distance from source to target is l in traditional Dijkstra, each of the search will be having a complexity $O(b^{l/2})$ and the total search time of bidirectional search will be much less than $O(b^l)$ in traditional Dijkstra which is equivalent to O(n+m log n) with Fibonacci heaps. If each queue operation takes O(n+m log n) time, the expected running time is $O(\sqrt{n}$+m log n).

The parallel version of the above algorithm has parallelized initialization phase and update phase. This parallelization is achieved using OpenMP constructs in the for loops. Depending on the number of processor run in the system, the iterative operations will be parallelly executed using multithreaded approach. Each process P_i (1<=i<=k) (k being the number of processors in the system) initializes the forward priority queue Q and reverse priority queue rQ partly. For large value of n, the operations of initialization phase and update phase were shared by multiple processors and which takes a time t which is less than the time achieved in sequential bidirectional search. If the node size is very less the time for communication between threads is eaten by the computation time. Even then the worst case running time will be less than the time achieved in in sequential bidirectional search i.e part of O(n+m log n) or $O(\sqrt{n}$+m log n)

2.2 Parallelized Landmarks

With parallelized preprocessing, it is possible to gather information about the graph that can be used to obtain improved lower bounds. In [10], a small fixed sized subset $L \subseteq V$ of "landmarks" is chosen in parallel. Then, for all nodes $v \in V$, the distance d(v, l) to all nodes $l \subseteq L$ is precomputed and stored in parallel. The time for precomputation and storage is certainly less than O(v) in sequential preprocessing. These distances can be used to determine a feasible potential. For each landmark $l \in L$, a potential is defined. The potentials for L landmarks will be calculated in parallel which is less than the time required in sequential form of potential calculation i.e O(L), L being the number of landmarks.

Doing the shortest path computation using landmarks in Dijkstra reduces V – L nodes thereby the time O(V-L) will be reduced for each instance. Landmarks in the graph usually attract the search to themselves. Then the potential calculation process for new s-t path will be faster as $L \in V$. The total time for landmarked Dijkstra will be O(L+mlog L) where $L \in V$, a set of landmarks. Doing the same with multiple processors and sharing the preproccessing phase, landmark selection and initialization phase and update phase reduce the time by O(V-L). The overall time O(L+mlog L) will be shared by $\sum_{i=1}^{k} P_i$ processors.

2.3 Parallelized Multilevel Approach

As the graph is decomposed into l+1 levels it takes O(l+1) time to decompose the graph. The subgraph at each level i consists of Si nodes. If $Si \in V$ and the subgraph with Si selected nodes consists of N_{Si} nodes and MSi edges then Dijkstra's algorithm is run on this subgraph to construct the shortest path which takes a time T(i) of $O(N_{Si}+M_{Si} \log N_{Si})$ time. It proceeds for all levels and for all subgraphs $S_1,S_2,..S_i,\ldots S_l$. The total time will be the sum of all the $\sum_{i=1}^{L} T(i)$ plus time to find the shortest path for the new graph, which comes out of additional edges. This time will be less than the time achieved in traditional Dijkstra.

In parallel multilevel approach, especially in the preprocessing phase the decomposition is carried out in parallel. Assuming $\sum_{i=1}^{k} Pi$ processors are involved then each processor select Si nodes at each level i and form a subgraph Gs when L=K. The special edges between levels are constructed in parallel. If E be the number of additional edges added then it takes O(E) time to construct sequentially. When we do the same in parallel, k processors will share the job and it takes a time of greater than O(E/k) for each processor.

Table 1. Parallel regions in speedup techniques

Technique	sequential	parallel
Bidirectional	Initialization phase Node selection phase Update phase	Initialization phase - parallel Node selection phase Update phase – parallel
Landmark	Landmark Selection Djikstra : Initialization & node selection Updation	Landmark Selection - parallel Djikstra :Initialization & node selection Updation - Parallel
Containers	Container Construction Updation Djikstra :Initialization Updation	Container Construction - parallel Updation - parallel Djikstra :Initialization Updation - parallel
Overlay graphs	Overlay level assignment Djikstra :Initialization Updation	Overlay level assignment - parallel Djikstra :Initialization Updation - parallel
Arc flags	Assignflags :Initialization Updation Djikstra :Initialization Updation	Assignflags :Initialization - parallel Updation - parallel Djikstra :Initialization Updation - parallel

2.4 Parallelized Containers

The Geometric containers [14] help to reduce the search space of Dijkstra’s algorithm by enclosing a list of target nodes for each edge inside a geometric object. The geometric information associated with each edge is then used for improving the performance of shortest path computations. Let G= (V, E), w: E → R be a weighted graph. It is remembered that a set of nodes C⊆ V is called a container. A container C associated with an edge (u, v) is called consistent, if for all shortest paths from u to t that start with the edge (u, v), the target t is in C. In other words, C(u, v) is consistent, if S (u, v) ⊆ C(u, v), where S (u, v) represents the set of nodes x for which the shortest u-x-path starts with the edge (u, v). Note that further nodes may be part of a consistent container. However, at least the nodes that can be reached by a shortest path starting with (u, v) must be in C(u, v). The additional nodes are referred as wrong nodes, since they lead the search in the wrong way. The preprocessed information is used for determining if a particular edge will be present on a shortest path to a given target.

In this method it is advantages to have only a subset of the neighbors ‘v’ of a node ‘u’ (those which belong to C(u,v)) are visited and thereby the search space is reduced. Even the simple bounding boxes reduce search space by a reasonable percentage [14]. The locations are fit into mathematical model eases down computation efficiency. There are many strategies present for container creation. Choosing an appropriate strategy for container creation itself is a time consuming process. The quality of containers will be minimized when the opted container is very large. These issues also ensures from the mathematical models, where each shape requires a different mathematical model.

2.5 Parallelized Arc flags

Arc Flags [15] is one in which we will partition the node set in p regions with a function r: V –> {1,…, p}. Then an arc flag, i.e. a p-bit-vector where each bit represents one region is used as edge label. For an edge e, a region is marked in the p bit-vector of e if it contains a node v with v ∈ S(e). Then the overall space requirement for the preprocessed data is θ(p · m). But an advantage of bit vectors as edge labels is the insight that the preprocessing does not need to compute all-pairs shortest paths. Every shortest path from any node s

outside a region R to a node inside a region R has to enter the region R at some point. As s is not a member of region R, there exists an edge e = (u, v) such that r (u) ≠ r (v). It is therefore sufficient, if the preprocessing algorithm regards only the shortest paths to nodes v that are on the boundary of a region. These paths can be determined efficiently by a backward search starting at the boundary nodes. Usually, the number of boundary nodes is by orders of magnitude smaller than n. A crucial point for this type of edge labels is an appropriate partitioning of the node set. Using a layout of the graph, e.g. a grid, quad-trees or kd-trees is found to be a tough task.

The preprocessing method using arc flag vector helps to reduce the search space as there is no requirement to compute all-pair shortest path between the vertices of the graph. And also preprocessing requires only nodes present in the boundary of a region for shortest path computation. Because of this the execution time is reduced. Partitioning of the node set plays a vital role in this method. i.e., Purely depends on the partitioning strategy adapted. The additional space utilized for storing the arc flag vector is not always compromising.

3. EXPERIMENTAL RESULTS

The speedup techniques were experimented on a PC with Intel Core2Duo processor (2.83Ghz) with 4 GB RAM running Ubuntu 10.04. Library of Efficient Date types and Algorithms (LEDA)[28] was used for easy implementation of various data types such as graphs, lists, priority queues, arrays, etc. OpenMP[29] API Parallel programming constructs are used for parallelizing the program.

The main goal of this work is compare the speedup techniques with respect to the computation time of the algorithm and its number of nodes visited during the journey of shortest path. The metrics like speedup based on runtime of the algorithm and the number of vertices visited during shortest path computation, which evaluates the natural abstractions of the system were considered. The speedup techniques were implemented and tested for random and planar graphs with a node count of upto 50000. The random graph can be considered for network applications or as a model of real-world networks such as the Internet, social networks or biological networks. The planar graphs can be considered in applications like telecommunications - e.g spanning trees, Vehicle routing – e.g. planning routes on roads without underpasses. For the nodes ranging from 10000 to 50000, the runtime and vertices visited during shortest path computation are tabulated. Then the average runtime and average number of nodes visited for each speedup technique are recorded and subsequently the speedup was calculated.

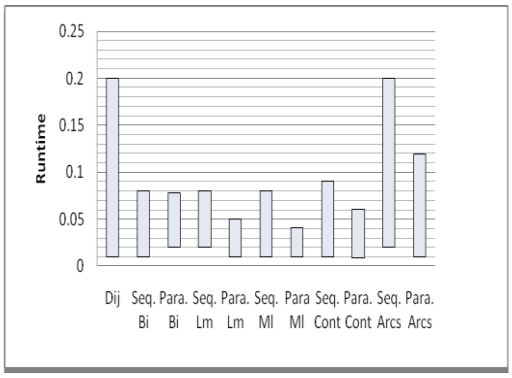

Fig 1: Comparison of runtime achieved by various speedup techniques in random graphs

The range of runtime for each speedup technique is charted in Figure 1. From the chart it is clear that parallelized speedup techniques show a reduction in runtime and also in its range. Better optimization is achieved with parallel processing constructs. The speedup achieved is the ratio of the performance of plain Dijkstra and the performance of Dijkstra with the specific speedup techniques applied. Running times are dependent on machine in which the algorithm is implemented and implementation strategy adapted.

Despite their demerits, running times are important for sanity-checking and complement efficiency to provide a better understanding of practical performance of algorithms under consideration. However, efficiency is closely correlated with running time. Parallelized containers yield a major performance improvement. The average runtime achieved in random graphs are promising and the results of the same are tabulated in Table 2.

Table 2. Runtime reduction in Random Graphs

	% of Average Runtime Reduction in Random Graphs
Landmarks	39%
Multilevel	47%
Containers	100%
Arc Flags	41%

When the arc flag initialization phase and update phase are parallelized, the average runtime is reduced by 41% compared to sequential arc flag method. In landmarks it gives a reduction in runtime of 39% while parallelizing the landmark selection and update phase. Overlays give a runtime reduction of 47% and containers gives 99.9% reduction in runtime. Bidirectional search as it takes time for construction of forward and reverse graphs the runtime is more for parallelized approach. At the outset, the preprocessing based methods are showing reduction in runtime.

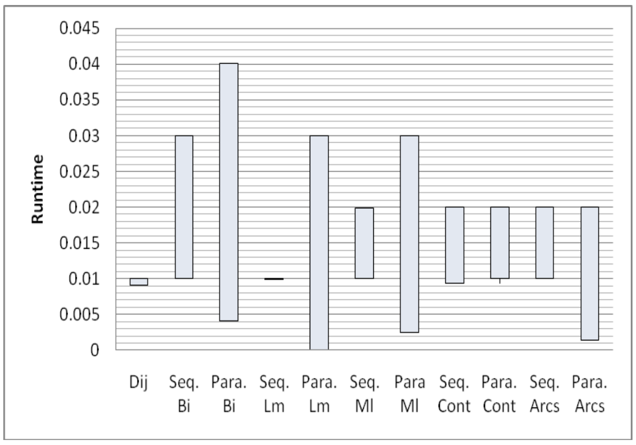

Fig 2: Comparison of runtime achieved by various speedup techniques in Planar Graphs

Planar Graphs belong to point to point network category. When the same speedup techniques and parallelization of the same is tested in planar graphs the results were worse. Only in arc flags, while parallelizing the flag initialization and update method a reduction in average runtime of 28% is achieved. In other techniques the intermediate node strength is reduced while constructing the planar graph itself.

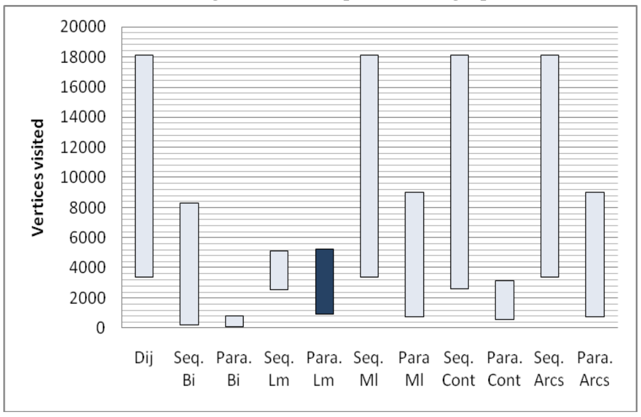

Fig 3: Comparison of Vertices visited achieved by various speedup techniques in Random Graphs

Another output metric is the average number of vertices visited during the shortest path computation is charted in Figure 3. In all, the performance is poor in parallelized landmarks as the edge weights have to be calculated in parallel during the search process. This additional effort usually increases the running time or vertices visited per visited edge by a little constant factor. Random graphs are showing good results while parallelizing the speedup techniques, which is tabulated in Table 3.

Table 3. Vertex Count Reduction in Random Graphs

	% of Average Runtime Reduction in Random Graphs
Bidirectional	89%
Landmarks	19%
Multilevel	54%
Containers	82%
Arc Flags	54%

Parallelizing bidirectional search in initialization phase and update phase shows 89% reduction in number of vertices visited, landmarks in landmark initialization and update phase gives 19%, multilevel method in overlay construction and update phase gives 54%, Containers in container construction and update phase gives 82% and arc flags in flag assignment and update phase give a reduction in percentage in vertices visited during shortest path computation of 54%

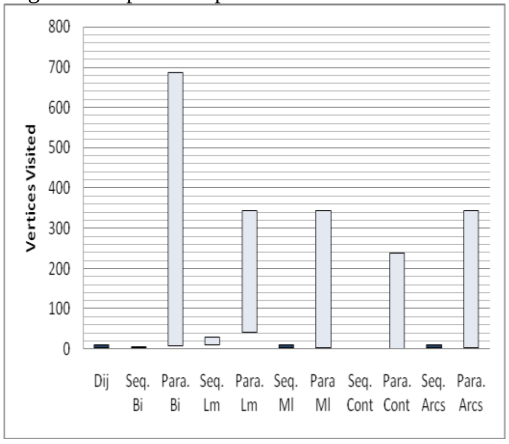

Fig 4: Comparison of Vertices visited achieved by various speedup techniques in Planar Graphs

The performance of parallelization is very poor in planar graphs. As the graphs have point to point connections the exact connection between the sites will be comparatively less than that of random graphs. Hence it takes a longer time to find the shortest path and also the number of nodes visited.

The speedup achieved is the ratio of the performance of plain Dijkstra and the performance of Dijkstra with the specific speedup techniques applied. The speedup was measured by considering the system computation time and vertices visited during shortest path computation in mind and by comparing the speedup technique considered with Dijkstra'a runtime. The speedup is measured for both sequential and parallelised speedup techniques. The relative performance gain in parallelization is the percentage of increase in speedup of the system. The percentage of increase or decrease in speedup when parallelizing the speedup techniques is measured for random and planar graphs and is recorded in the table Table 4.

TABLE 4. PERFORMANCE OF PARALLELIZATION FOR SPEEDUP WITH RESPECT TO RUNTIME

	% in Random	%in Planar
Arc flag	69.7270567	40.6140145
Container	45.7674365	-2.3253493
Bi-dir	-8.55313961	-9.59731999
Landmark	65.0182334	-33.6062573
Overlay	90.9976444	-7.66501065

The performance gain is high in random graphs than planar graphs as the update phase takes less time in random graphs than planar graphs due to its internal graph structure. In bidirectional search it shows a decrease in gain as the processing of shortest path computation switches between forward graph and reverse graph.

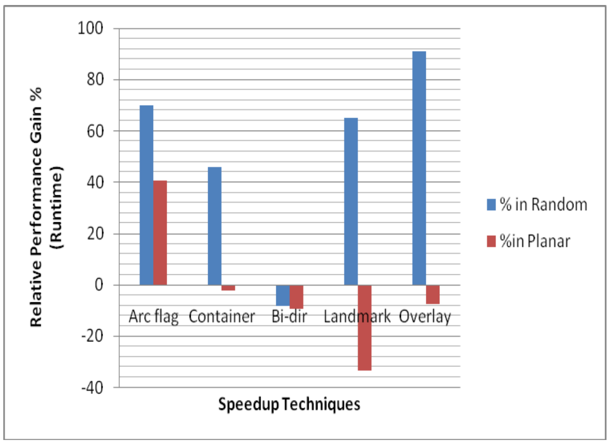

Fig 5: Performance gain in Speedup with respect to runtime

In planar graphs, except arc flags the other speedup techniques show a decrease in performance gain due to its internal structure. Arc flags show good results both in random(69%) and planar graphs(40%). It is because of the advantage of shortest path search space pruning effect of arc flag technique due to flag vectors.

TABLE 5. :PERFORMANCE OF PARALLELIZATION FOR SPEEDUP WITH RESPECT TO VERTEX COUNT

	% in Random	%in Planar
Arc flag	120.167981	-95.9654179
Container	456.033324	-99.1631799
Bi-dir	830.567686	-98.70317
Landmark	24.2957173	-89.3229167
Overlay	120.167981	-95.9654179

The percentage of increase or decrease in speedup with repect to vertices visited when parallelizing the speedup techniques is measured for random and planar graphs and is recorded in the table Table 5. The best performance is achieved in random graphs than planar graphs with respect to vertices visited during shortest path computation. Here bidirectional search(830%) gives very high performance gain as the two searches narrow down the search process fastly in random graphs.

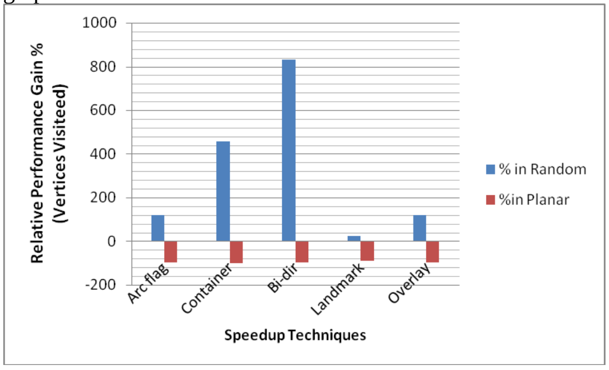

Figure 6. Performance gain in Speedup with respect to runtime

It shows always in drecrease in performance gain for speedup with respect to vertices visited as the planar graph structure plays an important role. Parallelization will not provide a good impact in planar type of netwroks. For these types of networks it is suggested to have the seqeuntial version of the speedup technique than parallelized version.

4. CONCLUSION

The performance of the speedup techniques for Dijkstra's algorithm like Bidirectional Search, Goal directed Search,based on landmarks, Arcflags, Containers and Multilevel technique were implemented in sequential and parallel using muticore architecture and parallel programming constructs and the results were compared and analysed. Important metrics for evaluation of the techniques like average percentage of reduction in runtime and the average number of vertices visited during shortest path computation were considered and compared. Additionally relative performance gain in parallelization is recorded for random and planar graphs with respect to runtime and vertices visited. Arc flags provide good results in random and planar graphs with respect to runtime and the number of vertices visited during shortest path computation. In general planar graphs give a gain in percentage for arc flags in runtime and in the rest it shows poor results for performance gain.
The performance of the system can be verified in real world graphs like time table information systems, etc., As arc flags give better results the same can be combined with other speedup techniques in dynamic time dependent networks.

5. REFERENCES

[1] DIJKSTRA, E. W. (1959) 'A note on two problems in connection with Graphs', In Numerische Mathematik, Vol. 1, Mathematisch Centrum, Amsterdam, The Netherlands, pp.269–271.

[2] Frank Schulz, Dorothea Wagner, and Weihe, K. (2000) 'Dijkstra's algorithm on-line: An empirical case study from public railroad transport', ACM Journal of Experimental Algorithmics, Vol. 5.

[3] I. Phol. (1971) 'Bi-directional Search', In Machine Intelligence, volume 6, pp 124-140. Edinburgh Univ. Press, Edinburgh

[4] Andrew V.Goldberg and Chris Harrelson. (2005) 'Computing the Shortest Path: A* Search Meets Graph Theory', In Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms.

[5] Andrew V. Goldberg and Renato F. Wernecky. (2005) 'Computing Point-to-Point Shortest Paths from External Memory', In Proc. Of The Seventh Workshop on Algorithm Engineering and Experiments (ALENEX05).

[6] Frank Schulz, Dorothea Wagner, & Christos Zaroliagis. (2002) 'Using multi-level graphs for timetable information in railway systems', In Proc. 4th Workshop on Algorithm Engineering and Experiments. LNCS 2409, Springer-Verlag, New York. pp43- 59.

[7] Sanders, P. and Schultes. D. (2005) 'Highway hierarchies hasten exact shortest path queries', In the Proceedings European Symposium on Algorithms.

[8] Sanders, P. and Schultes, D. (2006) 'Engineering highway hierarchies', In the Proceedings of the 14th European Symposium on Algorithms. LNCS,vol. 4168. Springer, New York. Pp.804–816.

[9] Schultes. D and Sanders. P. (2007) 'Dynamic highway-node routing', In Proceedings of the 6thWorkshop on Experimental and Efficient algorithms,LNCS. Springer, New York pp.66–79.

[10] Martin Holzer, Frank Schulz and Dorothea Wagner. (2008) 'Engineering Multilevel Overlay Graphs for Shortest-Path Queries', ACM Journal of Experimental Algorithmics, Vol.13, Article No.2.5, September.

[11] Dorothea Wagner and Thomas Willhalm. (2005)'Geometric Containers for Efficient Shortest-Path Computation', ACM Journal of Experimental Algorithmics, Vol.10, Article No.1.3, pp.1-30.

[12] Mohring, R. H., Schilling, H., Schutz, B.,Wagner. D., and Willhalm, T. (2006) 'Partitioning graphs to speed up Dijkstra's algorithm', ACM Journal of Experimental Algorithmics, Vol.11, Article No.2.8, pp.1-29.

[13] GUTMAN, R.J. (2004) 'Reach-based routing: A new approach to shortest path algorithms optimized for road networks', In Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics.

[14] Dorothea Wagner and Thomas Willhalm. (2007) 'Speed-Up Techniques for Shortest-Path Computations', In Proc. STACS 2007, LNCS , Springer-Verlag, New York. pp43- 59.

[15] Holzer, M, Schulz. F, Wagner and Willhalm. T. (2006) 'Combining speed-up techniques for shortest-path computations', ACM Journal of Experimental Algorithmics, Vol.10, Article No.2.5, pp.1-18.

[16] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, & Dorothea Wagner. (2010) 'Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm', ACM Journal of Experimental Algorithmics, Vol. 15, Article No. 3.

[17] The Advanced Computing Systems Association (2000) 'Amdahl's law & Parallel Speedup', http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/brownrobert/brownrobert_html/node3.html

[18] Daniel Delling, Bastian Katz and Thomas Pajor, "Parallel Computation of Best Connections in Public Transportation Networks" , In: *24th International Parallel and Distributed Processing Symposium (IPDPS'10)*, pages 1-12., IEEE Computer Society, 2010.

[19] R. C. Paige and C. P. Kruskal, "Parallel algorithms for shortest path problems," 1985, pp. 553–556.

[20] J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan, "Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation," Comm. ACM, vol. 31, no. 11, pp. 1343–1354, 1988.

[21] K. M. Chandy and J. Misra, "Distributed computation on graphs: Shortest path algorithms," Comm. ACM, vol. 25, no. 11, pp. 833–837, 1982.

[22] K. V. S. Ramarao and S. Venkatesan, "On finding and updating shortest paths distributively," J. Algorithms, vol. 13, pp. 235–257, 1992.

[23]Dominik Schultes, Johannes Singler, Peter Sanders. (2008)'Parallel Highway Node Routing', A Technical Report, Feburuary. algo2.iti.kit.edu/schultes/hwy/parallelHNR.pdf

[24]R.Kalpana, P.Thambidurai, Arvind Kumar, R. Parthasarathy, and Praful Ravi. (2010), 'Exploiting Parallelism in Bidirectional Dijkstra for Shortest-Path Computation', in the Proceedings of International conference on Computers, Communication and Intelligence at , Vellammal college of Engg., & Tech.,Madurai, India, pp. 351-356, July.

[25]R.Kalpana, P.Thambidurai,(2010), 'Optimization of Landmark preprocessing with Mulitcore Systems', Journal of Computing, Vol.2, Issue.8, pp.102-108, August.

[26]R.Kalpana, P.Thambidurai (2011), 'Optimizing shortest path queries with parallelized Arc flags', in the Proceedings of IEEE International conference on Recent trends in Information Technology, MIT Campus,Anna University, Chennai, India, June.

[27]R.Kalpana, P.Thambidurai (2011), 'Parallelized Multilevel Arc Flags Improve Speedup In Shortest Path Queries', , in the Proceedings of the IEEE International Conference on Process Automation, Control and Computing, CIT, Coimbatore, July 2011.

[28] 'The OpenMP - API specification for parallel programming', available at http://www.openmp.org

[29] Algorithmic Solutions **Software GmbH (1995)** 'LEDA', available at http://www.algorithmic-solutions.com