Land Forces Academy Review
Vol. XXIV, No 2 (94), 2019

STUDY OF PARALLEL ALGORITHMS FOR IP SWITCHES

Mhnd FARHAN
University of Baghdad, Iraq
mhndfarhan@yahoo.com

ABSTRACT

Owing to the increase in the internet traffic, any calculation that requires more than linear time would be considered unreasonably moderate for constant applications. One cure is to utilize numerous processors to build up associations in parallel and the other is to construct low cost, high speed, large capacity non-blocking switching architecture. In this paper, our focus is on developing parallel algorithms for routing which will ensure high-speed internet connectivity and at the same time making the system to be cost effective.

KEYWORDS: parallel algorithms, IP switch, Dijkstra algorithm, integrated development environment

1. Introduction

In the previous years, we have seen the Internet advancing from a niche network associating scholastics and research foundations to the fundamental worldwide data and excitement network. This is because of innovations that have encouraged generally transmission capacity accessibility and radically decreased expense of information exchange, making a perfect situation for the mass multiplication of the Internet. Because of such sensational advancement, existing applications, that have ordinarily been conveyed over various systems, are being re-created to be executed over the Internet (this is the situation for Voice over Internet Protocol (VoIP) and Internet Protocol Television (IPTV), while new applications keep on developing (peer-to-peer networking and interactive gaming). The increasing exploitation of the Internet has provoked an exponential increase in the bandwidth requirement on the underlying network architecture, a trend which is further emphasized by near-future applications (e.g. high definition IPTV and e-science applications) (Lee & Oruc, 1995).

Over the past decade, the increase in bandwidth requirement has been tackled by router vendors by increasing the speed of the electronic processing units, following the evolution predicted by Moore's Law. However, since the traffic exchanged in the Internet has grown at a much higher pace, there is a need to redesign the routing algorithms and implement in a manner which will facilitate reduction of internet traffic at no increase in cost (Maleki et al., 2016).

2. Design Methodology

A pseudo code is used to come up with a design of both a parallel and sequential design of Dijkstra algorithm in this section. In summary, to implement the pseudo code and come up with the two designs, a high level programming language (java) is used. The developed source code is run on Eclipse, an Integrated Development Environment (IDE) for Java. Note that the source code has classes for drawing the graph, the edge and vertices of sequential and parallel Dijkstra code (Singh & Khar, 2016). The configuration for the test code is then run and the output displays through the graphical user interface the shortest path between two vertices. In an actual network, this represent the shortest path between two nodes that is used by the data packets. More detail of this design is given in the sections below.

DOI: 10.2478/raft-2019-0019

2.1. System Design

It is in the design step that we choose how to partition the operations of our program into classes, we choose how these classes will cooperate, what information each will store, and what activities each will perform. The design solution of this paper comprises of three separate tasks namely; design of the sequential Dijkstra code, design of the parallel Dijkstra code and design of the graph to show case/display the output after the execution of the code. Each task is conceptually independent from the others as they are found in different classes, but each is necessary in order for the final program to be executed.

2.1.1. Flow Charts

A characteristic method to arrange different auxiliary segments of a software package is in a progressive style, which bunches comparative conceptual definitions together in a level-by-level way that goes from explicit to increasingly broad as one navigates up the pecking order. A typical utilization of such chains of command is in a flow chart. This sort of order is helpful in programming plan, for it bunches together regular usefulness at the most broad dimension, and perspectives particular conduct as an expansion of the general one. The flow charts in Figure no. 1 and Figure no. 2 encompassing the pseudo codes represent sequential and parallel Dijkstra algorithm respectively.

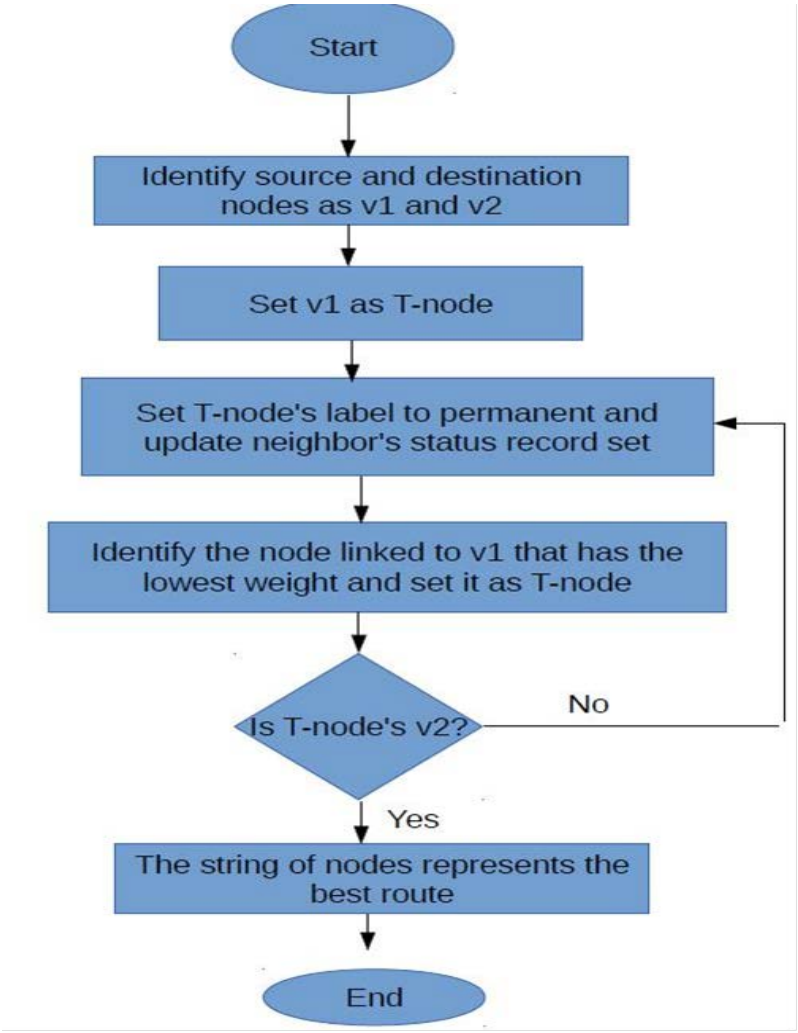

Figure no.1: *Flowchart for sequential Dijkstra algorithm*

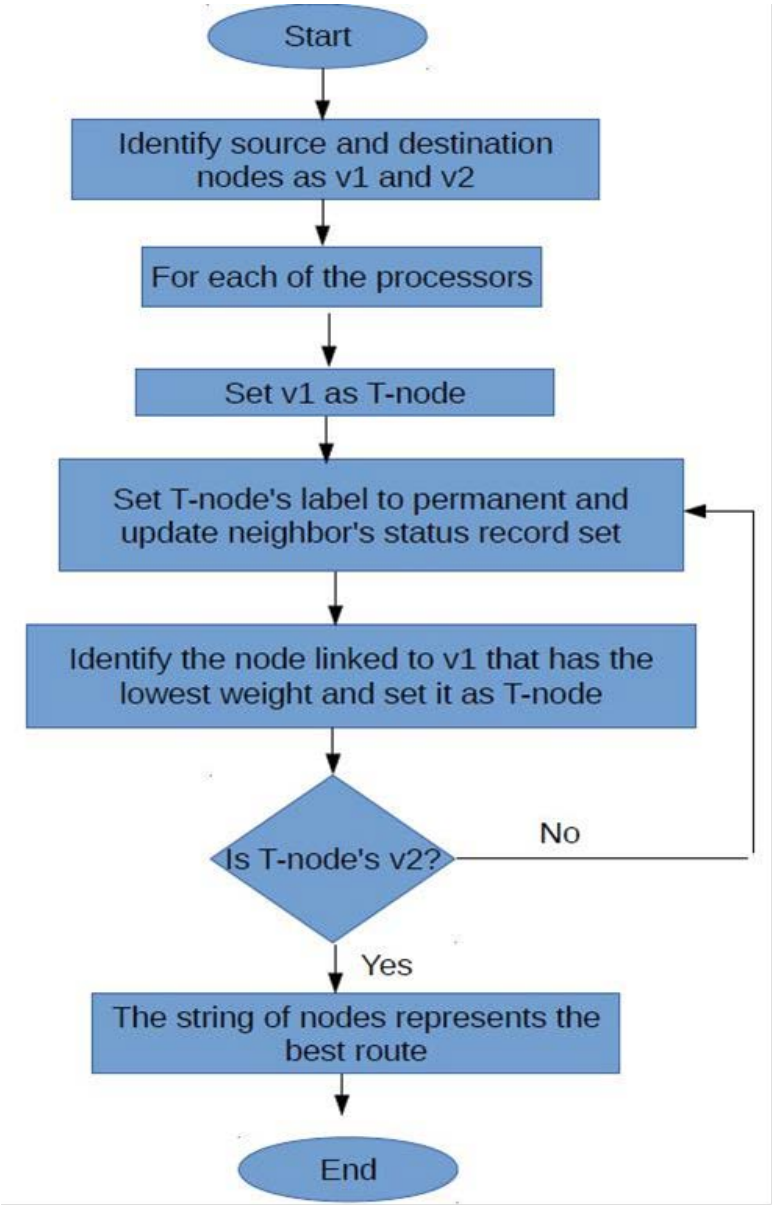

Figure no. 2: *Flowchart for parallel Dijkstra algorithm*

2.1.2. Programming

The programming language chosen to implement the design is Java 8 since Java is a concurrent programming language i.e. a programming language that use language constructs for concurrency. These constructs may involve multi-threading, support for distributed computing, message passing, and shared resources. The design of a Java program is determined by characterizing the classes, together with their instance variables and techniques. The flow chart graphs are utilized to express the association of the program; and the utilization of pseudo-code to portray the algorithms. These outlines are standard visual documentation used to express object-oriented software designs and act as a guide in programming.

Actual coding on a computer begins after decision has been made concerning the classes for this program and their responsibilities. For the Dijkstra algorithm, the classes that are made included the edge, vertex, graph, centre utility and a public class for Dijkstra algorithm (Prasad, Krishnamurthy & Kim, 2018). The actual code for the above named classes can be

created in this program by using either an independent text editor (such as emacs, WordPad, or vi), or the editor embedded in an IDE, such as Eclipse, Netbeans or Borland JBuilder. In this case, Eclipse, an integrated development environment is used. In order to visualize or represent the relationships between the different vertex and edges, we use a library known as jgraphx a java graph visualization library. This clearly depicts the interconnections of the network and allows one to set the source and destination vertices. It also contains a tab, ‖Get path‖ that allows the program to be executed thereby giving the shortest path between two vertices which is displayed below it for the parallel and sequential algorithm.

The design of the graph is as shown in Figure no. 3. It comprises 7 nodes (vertices) denoted by A, B, C, D, E & G. Each of the vertices is linked to another with edges that are weighted which can also be used to represent the distance. The Dijkstra algorithm embedded in the program will look for the shortest distance between two nodes and the time taken to reach the destination. This occurs for both the parallel and sequential execution.

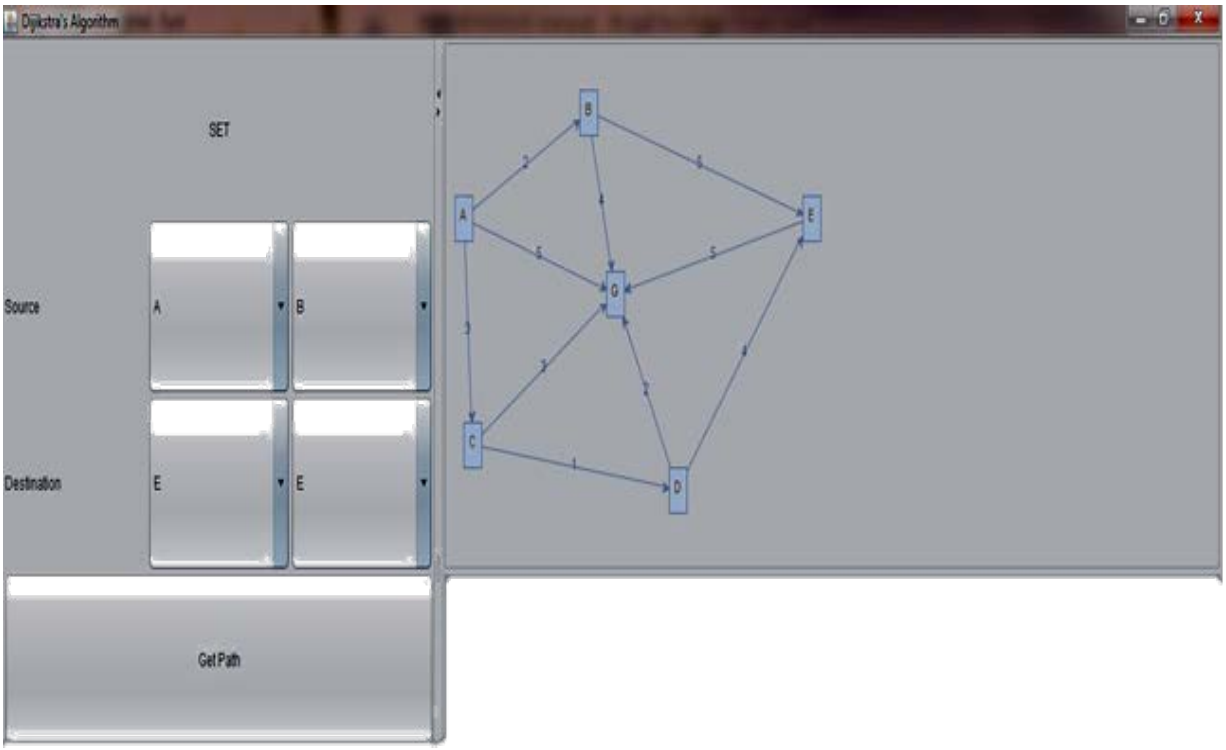

Figure no. 3: *Graphical representation of network design*

3. Results

After the execution of the program while interchanging the source and destination nodes, the obtained data detailing the source, destination, shortest path and time taken in nanoseconds is recorded as shown in Table no. 1.

Table no. 1
The Results

VERTICES (NODES)		**TIME TAKEN(nS)**	
SOURCE TO DESTINATION	SHORTEST PATH	SEQUENTIAL	PARALLEL
A-TO -E B-TO-D	A B E B G D	3910201	1969536
A-TO-D C-TO-E	A C D C D E	1855012	659720
B-TO-D C-TO-E	B G D C D E	635179	295936
A-TO-E C-TO-B	A B E C A B	1453694	878665
D-TO-B E-TO-A	D G B E B A	1026873	296417
D-TO-A E-TO-C	D C A E D C	1202991	284868

The speed-up factor for each result of the shortest path between two nodes is calculated as follows:

Speed- up Factor = (serial execution time/parallel execution time)

1. A-E & B-D S(p)= 3910201/1969536= 1.985341217
2. A-TO-D & C-TO-E S(p)= 1855012/659720=2.8118817135
3. B-TO-D & C-TO-E S(p)= 635179/295936=2.146363639073
4. A-TO-E & C-TO-B S(p)= 1453694/878665=1.654434853
5. D-TO-B & E-TO-A S(p)= 1026873/296417=3.464285112
6. D-TO-A & E-TO-C S(p)= 1202991/284868=4.222976958

4. Conclusion

Because of an expansion in parallelization of present day equipment, it is normal that parallel algorithms will assume progressively increasingly significant role in current processing. Shortest path algorithms, for example, the Dijkstra algorithm, assume a significant role in numerous pragmatic applications and enhancing it for different centers ought to bring progressively more advantages. This is the case especially in network routing since there has been an increase in demand for faster transmission rate and faster switching technologies due to the drive caused by an explosive growth of internet and an increase in traffic. From the foregoing, we have been able to achieve the objectives of the paper. We have investigated the use of parallel algorithms in improving switch throughput and come up with a suitable programming language with which to embed Dijkstra algorithm. Furthermore, as our experiment as demonstrated, we have been able to show comparison between parallel and standard (sequential) Dijkstra algorithm verifying the efficiency of the parallel one.

REFERENCES

Lee, C., & Oruc, A. (1995). A fast parallel algorithm for routing unicast assignments in Benes networks. *IEEE Transactions on Parallel and Distributed Systems, Vol. 6, Issue 3*, 329-334.

Maleki, S., Nguyen, D., Lenharth, A., Garzarán, M., Padua, D., & Pingali, K. (2016). DSMR: A parallel algorithm for single-source shortest path problem. *Proceedings of the International Conference on Supercomputing, Turkey*.

Prasad, A., Krishnamurthy, S., & Kim, Y. (2018). Acceleration of Dijkstra's algorithm on multi-core processors. *International Conference on Electronics, Information, and Communication, USA.*

Singh, D., & Khar, N. (2016). Modified Dijkstra's algorithm for dense graphs on GPU using CUDA. *Indian Journal of Science and Technology, Vol. 9, Issue 33*, 1-9.