(An Association Unifying the Sciences, Engineering, and Applied Research)
ISSN (Online): 2279-0055

International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Verilog Implementation of a System for Finding Shortest Path by Using Floyd-Warshall Algorithm

[1]Sachin Awasthi, [2]Vijayshri Chaurasia, [3]Ajay Somkuwar
Department of Electronics and Communication Engineering
Maulana Azad National Institute of Technology, Bhopal
INDIA

Abstract: *There are several applications in VLSI technology that require high-speed shortest-path computations. The shortest path is a path between two nodes (or points) in a graph such that the sum of the weights of its constituent edges is minimum. Floyd-Warshall algorithm provides fastest computation of shortest path between all pair of nodes present in the graph. With rapid advances in VLSI technology, Field Programmable Gate Arrays (FPGAs) are receiving the attention of the Parallel and High Performance Computing community. This paper gives implementation outcome of Floyd-Warshall algorithm to solve the all pairs shortest-paths problem for directed graph in Verilog.*

Keywords: *shortest path; vertex; all pairs shortest-paths; Floyd-Warshall algorithm ; FPGA; Xilinx ISE; Verilog; running efficiency.*

I. Introduction

In graph theory, the shortest path problem is the problem of computing a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized. Shortest path algorithms are applied to many fields, such as operations research, plant and facility layout, robotics, transportation, logistics, VLSI design etc [2]. There are many shortest path finding algorithms: the most widely used are Dijkstra, Floyd-warshall and bellman ford algorithms, all the three algorithms having different area of applications. For finding shortest path between any one pair of nodes in the graph Dijkstra algorithm is best suitable, for finding shortest path from one node to remaining all other nodes in a graph Bellman Ford algorithm is preferred and for computation of shortest path between all possible node pairs in the graph Floyd-Warshall algorithm (FWA) is fastest among all. The all-pairs shortest paths (APSP) problem is to find shortest paths between all-pairs of vertices in a weighted graph, and it is usually equivalent to make a table of minimum distances for every pair [3]. The problem is one of the most fundamental graph problems. We can find applications of the APSP problem in bioinformatics, social networking, traffic routing, etc. Moreover, the APSP problem can be generalized to the algebraic path problem [4-5] which covers several basic graph problems. A well-known solution of the APSP problem is to apply FWA algorithm which requires $O(n^3)$ operations on $O(n^2)$ memory space, where $'n'$ is the number of vertices in the graph.

Applications of Shortest-path algorithm-

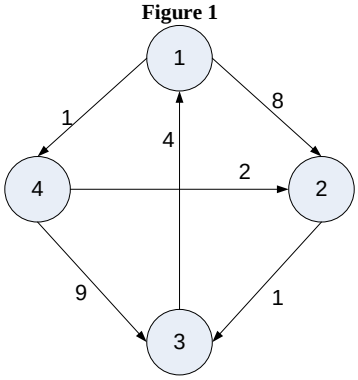
- Maps
- Robot navigation.
- Texture mapping.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Subroutine in advanced algorithms.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Network routing protocols (OSPF, BGP, and RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

II. Floyd-Warshall algorithm

Floyd-Warshall algorithm is a dynamic programming formulation, to solve the all-pairs shortest path problem on directed graphs[1]. It finds shortest path between all nodes in a graph. The algorithm considers the intermediate vertices of a simple path are any vertex present in that path other than the first and last vertex of

that path. Before 1960 it was tough task to find all pair shortest path. Warshall was interested to determine shortest path between each pair of vertices *u* and *v*, whether *u* can reach *v*. The Floyd-Warshall algorithm is the improvement of this algorithm, running in $O(n^3)$ time. The intelligence of the Floyd-Warshall algorithm is in finding a different formulation for the shortest path sub-problem than the path length formulation. At first the formulation may seem most unnatural, but it leads to a faster algorithm [9]. We will compute a set of matrices whose entries are $d_{i,j}^{(k)}$. We will change the meaning of each of these entries. For a path $p = (v_1, v_2, \cdots, v_n)$ we say that the vertices $(v_1, v_2, \cdots, v_{n-1})$ are the intermediate vertices of this path. Note that a path consisting of a single edge has no intermediate vertices. We define $d_{i,j}^{(k)}$ to be the shortest path from i to j such that any intermediate vertices on the path are chosen from the set$(1, 2, \cdots, k)$. In other words, we consider a path from i to j which either consists of the single edge(i, j), or it visits some intermediate vertices along the way, but these intermediate vertices can only be chosen from$(1, 2, \cdots, k)$. The path is free to visit any subset of these vertices, and to do so in any order. Thus, the difference between Floyd's formulation and the previous formulation is that the superscript (k) restricts the set of vertices that the path are allowed to pass through, and the superscript (m) restricts the number of edges the path allowed to use. For example, in the digraph shown in figure 1, notice how the value of $d_{3,2}^{(k)}$ changes as k varies. We compute $d_{i,j}^{(k)}$ assuming that we have already computed the previous matrix$d^{(k-1)}$. As before, there are two basic cases, depending on the ways that we might get from vertex i to vertex$j$, assuming that the intermediate vertices are chosen from $(1, 2, \cdots, k)$.

Figure 1



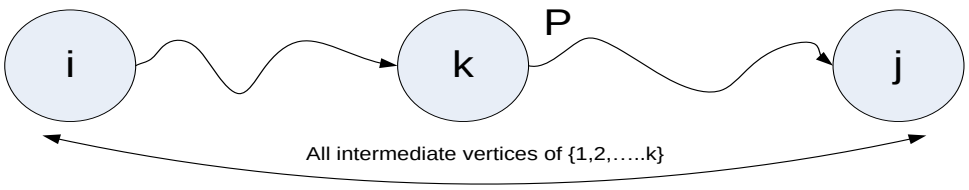
$$d_{3,2}^{(0)} = INFINITE(no\ path)$$
$$d_{3,2}^{(1)} = 12(3,1,2)$$
$$d_{3,2}^{(2)} = 12(3,1,2)$$
$$d_{3,2}^{(3)} = 12(3,1,2)$$
$$d_{3,2}^{(4)} = 7(3,1,4,2)$$

Any graph (directed or undirected) $G = (V, E)$ where V is the no. of vertices and E is the no. of edges. with weight function w: $\boldsymbol{E \rightarrow R}$ find for all pairs of vertices $u, v \in V$ the minimum possible weight for path from u to v.

- Finding the shortest path between all pair of vertices in a graph.
- For the general case negative edge is also allowed. The problem can be solved by running the Bellman-Ford algorithm 'V' times, once for each vertex as the source.
  Time complexity: $O\ (V^2E)$. If the graph is dense: $O\ (V^4)$
- The Floyd-Warshall algorithm solves the same problem in $O(V^3)$
  - Competitive for dense graph.
  - Use adjacency matrices, as opposed to adjacency list.
  - Let the vertices of G be $V = \{1, 2 \ldots n\}$ and consider $\{1, 2 \ldots k\}$ be a subset of vertices for some k.

For any pair of vertices $i, j \in V$ consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2 \ldots k\}$ and let P be a minimum weight path among them. The Floyd-Warshall algorithm exploits a relationship between path P and a shortest path from i to j with all intermediate vertices in the set$\{1, 2 \ldots k\}$. The relationship depends on whenever or not K is an intermediate vertex of$P$.

Figure 2 graph showing intermediate path



From figure 2 If K is not in$P$, then the shortest path from i to j with all intermediate vertices in the set $\{1, 2 \dots k\}$ is also a shortest path in the set$\{1, 2 \dots k\}$. If K is in$P$, then we break down P into P_1 and P_2 where,

- P_1 is the shortest path from i to K with all intermediate vertices in the set $\{1, 2 \dots k\}$.
- P_2 is the shortest path from K to j with all intermediate vertices in the set $\{1, 2 \dots k\}$.

III. Methodology

The methodology of Floyd-Warshall algorithm to find all pairs shortest paths is as follow:

Let $G = (V, E)$ be a weighted directed graph with an edge-weight function $w: E \to R$ that assigns real-valued weights to edges, where$V = \{1, 2 \dots n\}$ is a set of n vertices, and $E \subseteq V \times V$ is a set of edges. We assume that there are no negative-weight cycles in a graph G though the presence of negative-weight edges is allowed. Let us use an adjacency matrix representation for a graph$G$. The input of the all-pairs shortest paths problem is an $n \times n$ matrix $W = [w(i,j)]$ where $w(i,j)$ represents the edge weight from a vertex i to a vertex j in a graph G with n vertices, i.e.

$$w(i,j) = \begin{cases} 0 & if\ i = j; \\ weight\ of & if\ i \neq j\ and\ (i,j) \in E; \\ \infty & if\ i \neq j\ and\ (i,j) \notin E; \end{cases} \quad \dots\dots\dots\dots\dots\dots (1)$$

Let a path p be an arbitrary sequence of vertices. The distance of a path is the sum of lengths of all intermediate edges in the sequence of vertices. If we denote a set of paths from u to v with$P(u, v)$, the shortest-path distance between any pair (i, j) of vertices can be defined as

$$d_{(i,j)} = \min_{p \in P(i,j)} \{w(i, \mathrm{k}_1) + w(k_1, k_2) + \cdots + w(k_m,\ j)\} \quad \dots\dots\dots\dots\dots\dots (2)$$

Where m is the number of intermediate vertices. Then, the all-pairs shortest paths (APSP) problem is to find an $n \times n$ matrix $D = [d(i,j)]$ for a given $n \times n$ matrix$W$. We can define a recursive formulation to compute the shortest-path distances. Let $d_{(i,j)}^{(k)}$ be the distance of a shortest path (i, j) in which all the intermediate vertices are in the set $\{1, 2 \dots k\}$. the recurrence is defined as

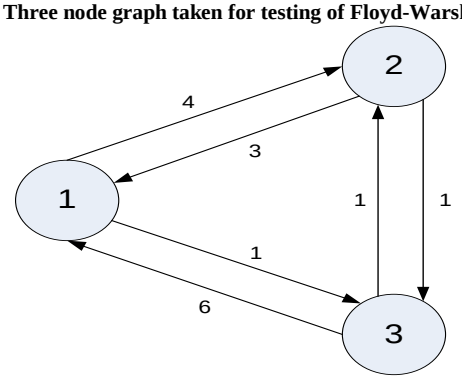
$$d_{(i,j)}^{(k)} = \begin{cases} w(i,j) & if\ k = 0; \\ \min\left(d^{k-1}(i,j), d^{(k-1)}(i,k) + d^{(k-1)}(k,j)\right) & if\ k \geq 1. \end{cases} \quad \dots\dots\dots\dots\dots\dots (3)$$

The matrix $D^n = [d^n(i,j)]$ gives an answer to the APSP problem, i.e. $D^n = D = [d(i,j)]$.

IV. Implementation and result

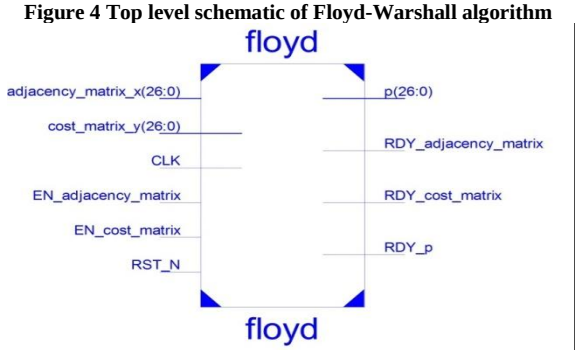
The Floyd-Warshall algorithm is implemented in Verilog by using Xilinx-13.2. By using this we can find all pairs shortest path in any given graph having 'n' no. of nodes. The simulation results are given by implementing it on graph shown in figure I.

Figure 3 Three node graph taken for testing of Floyd-Warshall algorithm



The graph is having three nodes in which we have computed all pairs shortest path. There are two input matrix of size 3×3, one is cost matrix and other is adjacency matrix. In the cost matrix the actual weight is written and in the adjacency matrix the connection is indicated whether the pair is connected or not. If connected then write '1' if not connected then '0'. Here one weight is indicated by three bits, so there will be maximum 27 bits.

Figure 4 Top level schematic of Floyd-Warshall algorithm



The top level schematic diagram of Floyd-Warshall generated through implementation in Verilog is shown in figure 4. It consists of six inputs and four outputs. The verilog has generated a pin diagram which defines input, output, and clock etc. The top level schematic in figure 4 consists of 10 pins each pin having its own working. Here input is taken as matrix i.e. adjacency matrix and cost matrix in adjacency matrix we define the direct connection between two nodes whether these are connected or not and in cost matrix the actual weight is written. The CLK pin is used to give clock. The EN_adjecency matrix and EN_cost matrix pin is given to enable the matrices for operation. RST pin is to reset the previous result. Here the output will be taken by the pin p(26:0) which is also in the matrix form. The final result obtained is shown in figure III.

Figure 5 Input-output waveform by simulation in verilog

Here the generated output waveform showing the output of each pin of pin diagram shown in figure II. Initially no input was there. After 400ns when input is applied in pin graph_in then corresponding output with clock will be generated this is shown in figure III.

V. Timing Detail

While finding shortest path the timing is main aspect and it should be as low as possible. The time taken is directly proportional to the total no. of nodes there in the graph. Here we have taken 3 nodes graph and the timing detail of that graph is given below. All values are displayed in nanoseconds (ns)

Timing constraint: Default OFFSET IN BEFORE for Clock 'CLK'
Total number of paths / destination ports: 757 / 22
Offset: 6.942ns (Levels of Logic = 5)

In the graph we have to move from source to destination. Here the source is cost matrix and the destination is 'p'. The total time is taken in the process is$6.942ns$. Here the table given below defines the separate time consume by each element and then total time.
Some Look Up Tables are generated in the architecture the processing time of each LUT is given in the table I

Table I Timing detail

Gate Net Cell: in->out	fanout	Delay	Logical Name (Net Name)
IBUF: I->O	2	1.218	0.622 cost_matrix_y_4_IBUF (cost_matrix_y_4_IBUF)
LUT3:I0->O	1	0.704	0.424 p_cmp_eq00011_SW0 (N5)
LUT4:I3->O	2	0.704	0.622 p_cmp_eq00011 (N1)
LUT3:I0->O	8	0.704	0.932 p_cmp_eq000130 (p_cmp_eq0001)
LUT3:I0->O	1	0.704	0.000 p_mux0000<20>11 (p_mux0000<20>1)
FDS: D		0.308	p_6
Total	6.942ns (4.342ns logic, 2.600ns route) (62.5% logic, 37.5% route)		

The table II gives the estimated value of Device Utilization. This table gives the total utilization of available devices i.e. no. of slices, no. of look-up tables (LUTs), and no. of input output buses and in terms of clock used with their % used

Table II Device Utilization summary (estimated value)

Logic utilization	Used	Available	% Utilization
Number of slices	19	4656	0%
Number of 4 input LUTs	33	9312	0%
Number of bonded IOBs	100	232	43%
Number of GCLKs	1	24	4%

All pair shortest path may also be computed by Dijkstra and Bellman Ford algorithms but these algorithms are the slower than that of Floyd-Warshall algorithm. Dijkstra algorithm is very much useful for finding the shortest distance between any two nodes. If we want to find all pairs shortest paths than Dijkstra algorithm need to run separately for all possible pairs in the graph. This process will consume large time. The Bellman Ford algorithm is used to find the distance from one source to remaining node. To find all pairs shortest path by Bellman ford algorithm needs to run separately for all nodes present in the graph. The Floyd-Warshall algorithm is the fastest algorithm for computation all pair's shortest paths in the graph, it provides all pairs shortest paths by running only once and is the fastest one.

VI. Conclusion

In this paper we have presented implementation of Floyd-Warshall algorithm for finding all pairs shortest paths in Verilog. To find all pairs shortest paths the Dijkstra algorithm need to run separately for each possible pair this process consume more time and is the slowest process. The Bellman Ford algorithm needs to run for each node and the Floyd-Warshall algorithm will provide all pairs shortest paths by running only once and is the fastest algorithm. The Verilog is the Hardware Description Language which can be used to generate hardware for any code. Verilog codes for Floyd-Warshall algorithm can be further used to burn Field Programmable Gate Array (FPGA). Since FPGA based hardware for path solvers performs much faster as compared to general purpose processor. The Floyd-Warshall algorithm can be efficiently applied to the systems of large node

number and it leads to a significant improvement in running efficiency. Xilinx ISE design suit is used with device as SPARTEN3E XC3S250E for the work presented

References

[1] Weisstein, Eric. "Floyd-Warshall Algorithm" .*Wolfram MathWorld*. Retrieved 13 November 2009.

[2] Chen, Danney Z. "Developing algorithms and software for geometric path planning problems". ACM Computing Surveys 28 (4es): 18. December 1996

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to algorithms,3rd ed. Massachusetts, USA: The MIT Press,2009.

[4] "Lecture 12: Shortest paths ". Network Flows and Graphs. Department of Industrial Engineering and Operations Research, University of California, Berkeley. 7 October 2008.

[5] G.Rote. "Path problems in graphs," Computing Supplementum, vol. 7, pp 155-198, 1990

[6] K. Matsumoto and S. G. Sedukhin, "The Algebraic Path Problem on the Cell/B.E. Processor," The University of Aizu,Tech. Rep. 2010-002, 2010.

[7] Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L. Introduction to Algorithms (1st ed.). MIT Press and McGraw-Hill. "The Floyd–Warshall algorithm", pp. 558–565,2009.

[8] Weisstein,Eric."Floyd-Warshall Algorithm".Wolfram Math World.

[9] Larson,R.and Odoni,A. "Shortest paths between All Pairs of Nodes." Urban Operations Research.1981.

[10] Loerch, U "Floyd's Algorithm." Auckland, New Zealand: Dept. Computer Science, University of Auckland,2000.

[11] Pemmaraju, S. and Skiena, S. "All-Pairs Shortest Paths" and "Transitive Closure and Reduction."Computational Discrete Mathematics: Combinatorics and Graph Theory in Mathematica. Cambridge, England: Cambridge University Press, pp. 330-331 and 353-356, 2003.

[12] Michalis Potamias , Francesco Bonchi, Carlos Castilloetal. Fast shortest path distance estimation in large networks, Proceeding of the 18th ACM conference on Information and knowledge management, 2009.

[13] R. Bellman, Dynamic Programming. Princeton University Press, Princeton, New Jersey, 1957.

[14] Kazuya Matsumoto,Naohito Nakasato, and Stanislav G. Sedukhin Blocked "All-Pairs Shortest Paths Algorithm for Hybrid CPU-GPU System",IEEE International Conference on High Performance Computing and Communications,2011.

[15] Dijkstra E W. A note on two problems in connection with graphs. Numerische Math,1:269-271,1959