Hardware Architecture for Finding Shortest Paths

K. Sridharan*, T.K. Priya †, P. Rajesh Kumar ‡
*Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai - 36
Email: sridhara@iitm.ac.in
† SoftJin Technologies, Bangalore, India
‡ Shri Vishnu Engineering College for Women, Bhimavaram - 534202, India

***Abstract*—The computation of shortest path for a mobile automaton between two points in the plane is considered in this paper. An architecturally-efficient solution based on Dijkstra's algorithm is presented for this problem. Results of implementation in Xilinx FPGA are encouraging: the solution operates at approximately 46 MHz and the implementation for a graph with 64 nodes and 88 edges fits in one XCV3200E-FG1156 device.**

Keywords: Shortest Path, Dijkstra's Algorithm, Hardware Architecture, Field Programmable Gate Arrays

I. Introduction

A key task for robots operating in industrial and other environments is navigation from one point to another. Planning a route [1] for the robot is typically done to meet two objectives simultaneously: a feasible path as well as a shortest path. The shortest typically minimizes the total time taken, reduces the cost of transportation and minimizes the chances of collision with other objects.

Computation of shortest paths is a well-studied combinatorial optimization problem. Prior sequential algorithms include the Dijkstra's $O(n^2)$ algorithm (n being the number of nodes) for single-source (multiple destinations) shortest paths with all edge costs being non-negative, and the Floyd's $O(n^3)$ algorithm [2] for computing all-pairs shortest paths. There are also estimator-type algorithms. A popular one is the A^* $Algorithm$ to solve the one-to-one shortest path problem based on a heuristic approach [3]. Recently, a number of algorithms in the domain of *constraint programming* have been proposed for solution of various combinatorial problems [4], [5]. However, all these algorithms have largely been investigated for implementation on general-purpose uniprocessors [6].

There are, however, a number of situations that call for computation or updation of the shortest path on the fly by a device on the vehicle/robot. These include planning or replanning routes in a dynamic environment. Further, computing the shortest path in real time using compact on-board electronics is crucial for mobile systems. Our interest is in an FPGA-based solution since it provides extensive support for implementation of parallel algorithms. Further, FPGAs allow on-site reconfiguration and do not require the customers to pay for large Non-Recurring Engineering costs and purchase expensive mask sets.

Considerable research has been done on development of FPGA-based adaptive control strategies for motor control and other applications. A review of the state of the art of FPGA design methodologies with focus on industrial control system applications is presented in [7]. The authors in [8] present a detailed discussion of advanced features, design tools and application domains for FPGAs. A bilateral teleoperation system using an FPGA is described in [9]. An embedded DSP-based fuzzy controller for autonomous mobile robots is presented in [10]. To the best of our knowledge, there is no space and time-efficient algorithm or FPGA implementation for the shortest path problem.

Our interest is in shortest paths particularly for a mobile robot operating in an environment with multiple obstacles. We assume a graph such as the complete visibility graph or reduced visibility graph is available and present a hardware-directed algorithm for shortest path based on Dijkstra's algorithm. An efficient FPGA-based implementation is presented. A key aspect of this approach is incorporation of *parallelism* at the algorithm level along with careful *reuse* of hardware to achieve time and space efficiency in the implementation. The scheme presented avoids use of sophisticated data structures. The hardware implementation minimizes multiplications and other expensive operations. The FPGA-based solution operates at approximately 46 MHz and the design fits in one XCV3200E device for fairly large problem sizes.

The rest of this paper is organized as follows. In the next section, we present the formulation. Section III presents the new algorithm. Section IV describes the VLSI architecture. Implementation of the algorithm in FPGA is presented in section V. Conclusions are given in section VII.

II. Preliminaries

Dijkstra's algorithm is a widely used shortest path search technique. It finds the shortest path between one node s and several distinctive nodes in a graph, and in the process also extracts the minimum cost path from any given destination to the source node [11].

The input to the algorithm consists of a directed graph $G = (V, E)$ and a positive weight c_i (representing distance) associated with each arc $e_i \in E$. We will denote by V the set of all vertices in the graph G. Each edge of the graph is an ordered pair of vertices (u,v) representing a connection from vertex u to vertex v. The set of all edges is denoted by E. An undirected graph can be converted to a directed graph by assigning two directed edges ab and ba for each edge ab.

Dijkstra's algorithm works by visiting vertices in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices

to the set of vertices to be examined. it expands outwards from the starting point until it reaches the goal.

The algorithm works by keeping for each vertex v, the cost $d[v]$ of the shortest path found so far from source vertex s. Initially, this value is 0 for the source vertex s and infinity (a very large value in practice) for all other vertices, indicating that we do not know any path leading to those vertices. When the algorithm finishes, $d[v]$ will be the cost of the shortest path from s to v or infinity, if no such path exists. The basic operation of Dijkstra's algorithm is edge relaxation: if there is an edge from u to v, then the shortest known path from s to u can be extended to a path from s to v by adding edge (u, v) at the end. This path will have length $d[u] + c(u, v)$. If this is less than $d[v]$, we can replace the current value of $d[v]$ with the new value. Edge relaxation is applied until all values $d[v]$ represent the cost of the shortest path from s to v. The algorithm is organized so that each edge (u, v) is relaxed only once, when $d[u]$ has reached its final value. The algorithm maintains a flag register ($flag[v]$) for each vertices, which is set to 1, if the shortest paths from the source have already been determined. The algorithm starts by assigning all flag registers to 0, and in each step one flag register corresponding to the minimum value of $d[u]$ is set to 1. When $flag[u]$ is set to 1, the algorithm relaxes every outgoing edge (u, v).

III. Proposed Algorithm

In this section, a parallel algorithm for computing the shortest path between a pair of distinct nodes using Dijkstra's algorithm is presented. Between each pair of adjacent nodes, it is assumed that the square of the distance is available (when coordinates of a node are integers, squaring results in integers which are digital-hardware friendly). As explained earlier, the algorithm gradually creates a tree from the vertices and some of the edges in the graph; the root of this tree is the source vertex (s). Thus, it requires proper maintenance of two arrays:
1. d[i] to store the shortest distance from source vertex to all other vertices i, ($i = 0, \cdots, n-1$),
2. path[i] to store the parent node or predecessor of each vertex i, and
3. flag[i] to store the set of vertices whose shortest paths from the source have already been determined.

ALGORITHM Find_Shortest_Path

Inputs: Total number of vertices/nodes (n), number of edges (m) and integer weight for each edge (square of distance between adjacent vertices is stored).

Outputs: The sequence of vertices (indices) determining the shortest path and the minimum cost.

Step 1: Initialize the shortest distance of vertices from source node (distance field) as $d[s] = 0$, where s is source node, and d[i] = FF (infinity) ($i = 0, \cdots, n-1, i \neq s$). Initialize min_node to s, flag[s] = 1, and flag[i] = 0, for ($i = 0, \cdots, n-1, i \neq s$).

Step 2: For $i = 0, \cdots, n-1$, calculate sum[i] **in parallel**, the sum of distance from min_node to the neighbouring nodes.

Step 3: Check if flag[i] = 0 and d[i] $\geq$ sum[i] for ($i = 0, \cdots, n-1$). If so assign path[i] = min_node and d[i] = sum[i].

Step 4: Find the index k for which d[i] is minimum using a binary tree based comparison unit.

Step 5: If k is the goal point, go to Step 6, else assign min_node = k and go to Step 2.

Step 6: Output the shortest path and minimum cost.

Remark 1: The complexity of the algorithm depends on the number of iterations required. In worst case, the solution requires n iterations. The complexity of each iteration can be analysed as follows. Steps 1, 2, 3 and 5 take constant time while Step 4 takes logarithmic time. Step 6 takes $O(n)$ time. Thus the overall complexity is $O(nlogn)$.

IV. Hardware Architecture

The proposed architecture is shown in figure 1. It consists of adder blocks, memory blocks, a binary tree-structured comparison unit and comparators. We have n blocks of memory. Each memory block has n locations of 8 bit data width. This helps to read n data items in parallel. The distance of edges from each vertex to all other vertices is stored in these blocks. If a vertex is not visible, FF_{Hex} is stored in the corresponding memory location. The shortest distance from start node to each vertex is stored in an array of registers (d). Two registers are used to store the $min\_distance$ value and min_node value. These are the outputs of the Binary-Tree Structured Comparison Unit shown in Figure 2. The $min\_distance$ register store the minimum distance value and min_node register store the index of minimum distance. Initially these register values are set to 0 and s (start node) respectively. Registers containing FLAG consists of n single bit registers. These registers are used to store the information about the set of vertices whose shortest paths from the source have already been determined. The parent node or predecessor of each vertex i is stored in Memory containing "path". The distances from min_node to the neighbouring nodes are calculated in parallel using n 8-bit adders (ADDER BLOCKS). The outputs of the adder blocks ($sum[i]$) are compared in parallel with the contents of d registers using n 8-bit comparators. The FLAG register values are used as a select signal. If flag[i] is 0, the comparator outputs the minimum value. If not, the comparator output is input d[i]. The d register values and Memory containing path are updated in parallel. The index (k) of minimum distance is calculated using binary tree-structured comparison unit. It uses n comparators and found the minimum value in $O(logn)$ time. The values of minimum distance and its index are stored in registers $min\_distance$ and min_node respectively. Also flag[k] is set to 1. The finish signal is set to 1, when min_node is same as end node. This finish signal is used as a select signal to output the shortest path.

V. FPGA Implementation

Results of VHDL code simulated using ModelSim followed by synthesis and implementation using Xilinx ISE are presented.

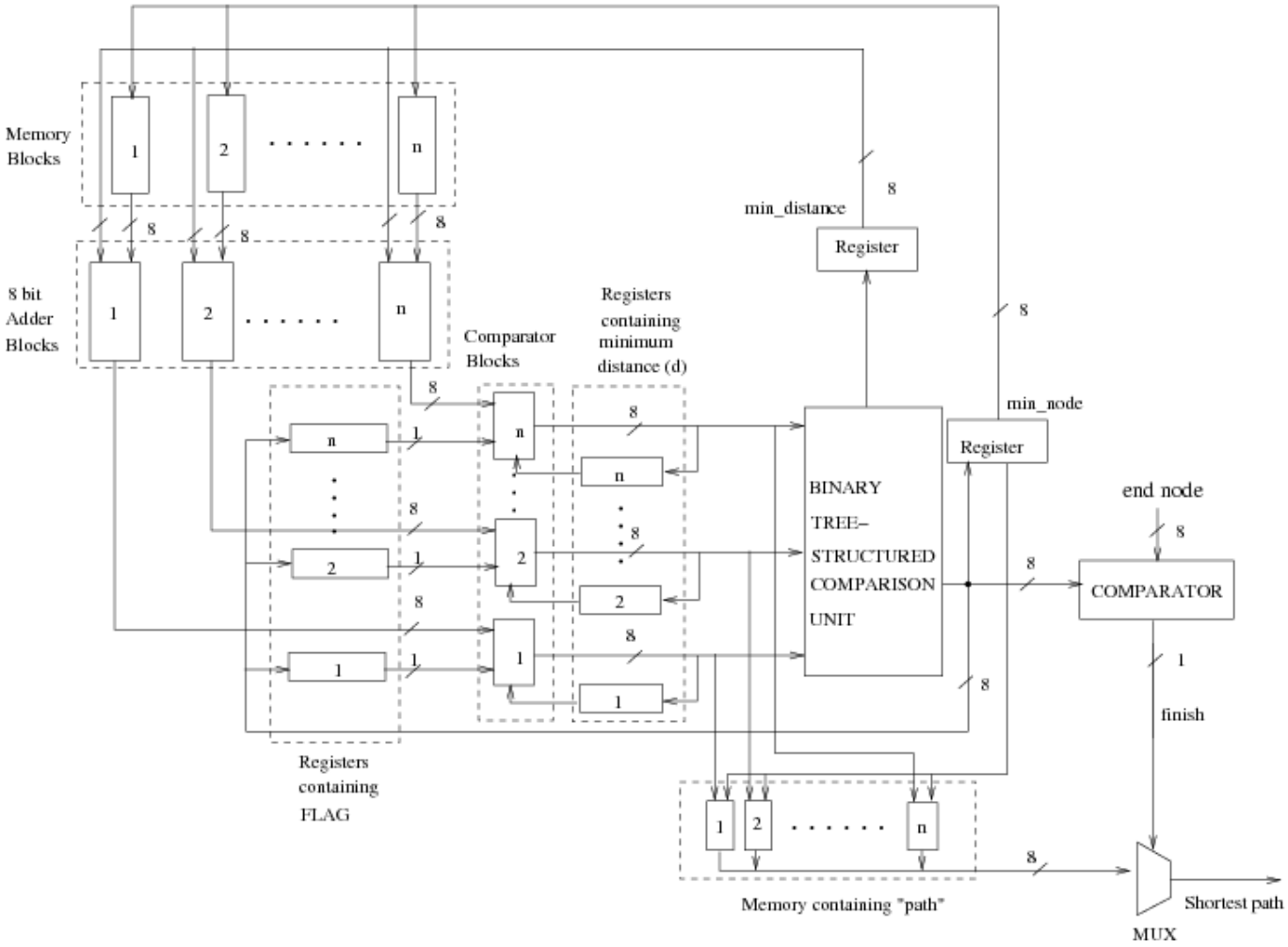

Fig. 1. Hardware for the proposed algorithm

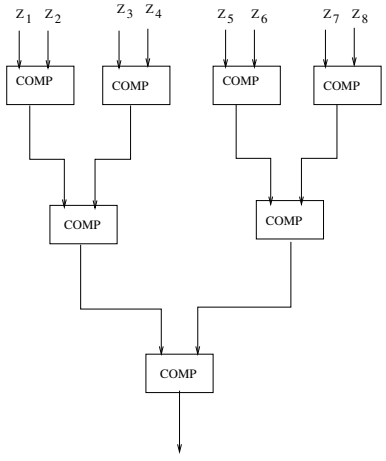

Fig. 2. Architecture for binary tree-structured comparison

TABLE I
EXECUTION TIME FOR DIFFERENT INPUT SIZES : IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

Environment (total vertex number)	Clock Rate (MHz)	Clock Cycles	Execution Time (μs)
48 vertices on XCV3200E-FG1156	45.6	300	6.6
64 vertices on XCV3200E-FG1156	45.6	328	7.2

In Table I, the results are for graphs with 48 and 64 vertices. Since there do not appear to be any other hardware implementations, information on the extent of speed-up obtained by our hardware implementation has been gathered by timing our C implementation (using the *times()* function) of the sequential version of the algorithm on a general-purpose PC with Pentium III CPU (1133 MHz) and on-board RAM of 512 MB. The PC runs Red Hat Linux 8.0. The CPU time taken for finding the shortest path for an environment with a total of 48 vertices is 0.24 milliseconds while for 64 vertices, it is 0.356 milliseconds. This amounts to a speed-up for the FPGA implementation by a factor of 50.

It can be observed that the design for an environment up to 64 vertices fits in one XCV3200E device.

The power consumption gathered using XPower (Xilinx tool) for different numbers of vertices is presented in Table III. The frequency given in Table I is the maximum frequency with which the hardware can operate. For power calculation, we have chosen a frequency (45.45 MHz), less than the maximum frequency which gives an integer value for the time period (22 ns). It is to be noted that the design of single-source shortest path finder for an environment with 64 nodes consume only 564 mW.

The floorplan for the implementation of Dijkstra's shortest path algorithm of an environment with 64 nodes and 88 edges on an XCV3200E is shown in Figure 3. The figure shows usage of various device elements as RAM/ROM etc.

VI. COMPARISON OF PROPOSED METHOD WITH AN APPROACH BASED ON LINEAR PROGRAMMING

A second method based on linear programming has also been implemented on FPGA. The linear programming approach is based on the following formulation:

Given a directed graph $G = (V, E)$ and a positive weight c_i (representing distance) associated with each arc $e_i \in E$, the calculation of shortest path can be expressed as the solution of the following linear program. We assume c represents a column vector made up of individual edge costs c_i.

Min $c^T X$
such that
$AX = b$,
$x_i \geq 0$

where x_i is a variable associated with each arc e_i to represent a flow of some imaginary commodity [12] through the arc in the direction of orientation and X is a column vector whose i^{th} component is x_i. The constraint set is built from node equations (for entering and leaving flows) written at each of the V nodes. The coefficient matrix A is obtained from the node equations. The proof that the above LP formulation indeed gives the cost of shortest path is as follows: the constraint equations together represent a flow of one unit leaving s and entering g. Flow conservation at each node is also expressed by the constraint equations. The elements in the summation in the objective function consider all arcs since the flow may, potentially, be through all of them. The minimization yields the least cost for the flow.

Our hardware directed solution is based on enhancements to the revised simplex method [13]. In particular, the entire A matrix is not stored. In the revised simplex method, a subset of A called as the initial basis matrix B is obtained and its inverse, denoted B^{-1} plays a crucial role. The calculation of vectors employing B^{-1} is done in parallel while we also make the observation that the constraints in the shortest path problem have a simple form with non-zero coefficients of X_i being only 1 or -1. Reuse of hardware is also a feature of our FPGA implementation. The implementation of single-pair shortest path using linear programming approach for an environment with 40 vertices fits into one XCV3200E (uses 99 $\%$ of Slices and LUTs). Further, the design has a maximum usable frequency of 72 MHz.

The proposed approach based on Dijkstra's algorithm turns out to be area-efficient in comparison to the LP-based solution since the implementation results for Dijkstra's shortest path algorithm show that the design upto 64 vertices fits into one XCV3200E device. However Dijkstra's does not directly solve the single-pair shortest path problem. The arithmetic operations required for Dijkstra's algorithm are only additions and comparisons. The linear programming approach requires cross-multiplication in addition to addition/subtraction and comparison.

VII. CONCLUSIONS

A hardware-directed solution to the shortest path problem based on Dijkstra's algorithm is presented in this paper.

Results of implementation on Xilinx FPGA device show high speed construction of shortest path.

REFERENCES

[1] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[2] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[3] E.P. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Cybern.*, SSC-4(2):100–107, 1968.

TABLE II
SPACE CONSUMPTION ON XILINX VIRTEX DEVICES : IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

Environment (Nature of objects)	Slices	Gate Count	Block RAMs	Bonded IOBs
48 vertices on XCV3200E-FG1156	16,386 (50 %)	1,809,145	96	58
64 vertices on XCV3200E-FG1156	32,446 (99 %)	2,481,514	124	58

TABLE III
POWER CONSUMPTION ON XILINX VIRTEX DEVICES : IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

Environment (Nature of objects)	Frequency (MHz)	Power Consumed (mW)
48 vertices on XCV3200E-FG1156	45.45	523
64 vertices on XCV3200E-FG1156	45.45	564

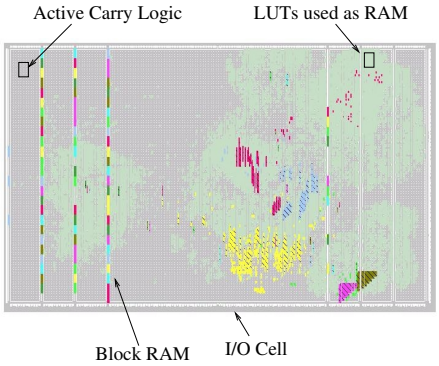

Fig. 3. Floorplan for the implementation of Dijkstra's shortest path algorithm for a graph with 64 nodes and 88 edges on an XCV3200E-FG1156

[4] R. Bartak and M. Milano (Editors). In *Proceedings of Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (LNCS 3524)*, 2005.

[5] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[6] H. Choset et al. *Principles of robot motion: Theory, algorithms, and implementations*. MIT Press, 2005.

[7] E. Monmasson and M.N. Cirstea. FPGA design methodology for industrial control systems - a review. *IEEE Transactions on Industrial Electronics*, 54(4):1824–1842, 2007.

[8] J.J. Rodriguez-Andina, M.J. Moure, and M.D. Valdes. Features, design tools, and application domains of FPGAs. *IEEE Transactions on Industrial Electronics*, 54(4):1810–1823, 2007.

[9] E. Ishii, H. Nishi, and K. Ohnishi. Improvement of performances in bilateral teleoperation by using FPGA. *IEEE Transactions on Industrial Electronics*, 54(4):1876–1884, 2007.

[10] J. Baturone, F.J. Moreno-Velo, V. Blanco, and J. Ferruz. Design of embedded DSP-based fuzzy controllers for autonomous mobile robots. *IEEE Transactions on Industrial Electronics*, 55(2):928–936, 2008.

[11] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[12] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Englewood Cliffs, 1982.

[13] V. Chvátal. *Linear Programming*. W.H. Freeman, 1990.