Parallel Implementation of Dijkstra’s Algorithm in FPGA

Ariel Faria (af2791), Michelle Valente (ma3360), Utkarsh Gupta (ug2121) and Veton Saliu (vs2519)
Department of Electrical Engineering
Columbia University in the City of New York, New York

Abstract

This report discusses our project’s design to speed up Dijkstra’s shortest path finding algorithm on FPGA (for the purpose of this project we’ll be using Altera’s Cyclone V board). Parallel implementation of Dijsktra’s algorithm on FPGA has been shown to overcome the downside of quadratic growth of the complexity of the algorithm with the number of nodes [1-3]. We will apply this algorithm to the specific application of solving for the shortest path through a maze.

Introduction

A. Dijkstra’s Algorithm

Dijkstra’s algorithm calculates the optimal path through a network, starting from the source node to a target node. In a maze solving application, the optimal path is the shortest path allowed between the entrance and exit, with the nodes characterized by branching points in a maze. The algorithm is as follows.

Let the value of each node correspond to the distance between the node and the initial node. The initial node is assigned a value of 0 and all other nodes are assigned the value of infinity. The value of each node adjacent to the initial node is compared to the distance between this and the initial node, and the smallest of these values is assigned to the node. We transition to the adjacent node with the smallest value. The same operation is performed, with the values at each yet unobserved node compared to the total distance of the node to the initial node, and the value of the node updated accordingly. These operations are repeated until the target node is reached. Pseudo code for Dijkstra’s algorithm, taken from Wikipedia [4], is shown below:

```
function Dijkstra(Graph, source):

    dist[source] ← 0                      // Distance from source to source
    prev[source] ← undefined              // Previous node in optimal path initialization

    for each vertex v in Graph:  // Initialization
        if v ≠ source            // Where v has not yet been removed from Q (unvisited nodes)
            dist[v] ← infinity           // Unknown distance function from source to v
            prev[v] ← undefined          // Previous node in optimal path from source
        end if
```

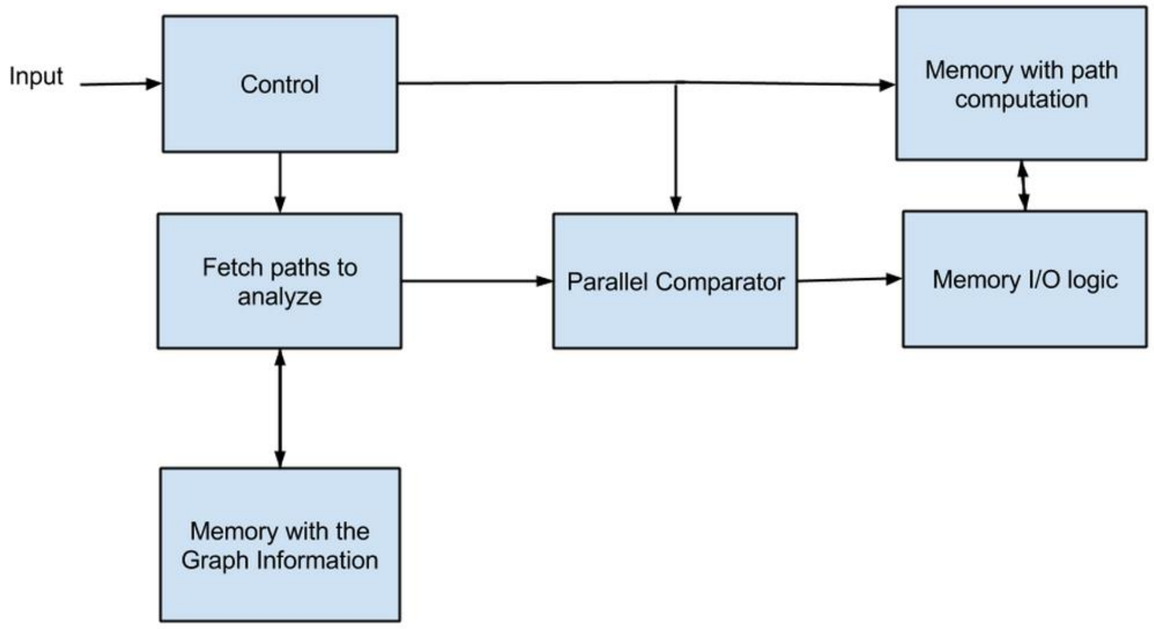
```
            add v to Q                        // All nodes initially in Q (unvisited nodes)
        end for

        while Q is not empty:
            u ← vertex in Q with min dist[u]    // Source node in first case
            remove u from Q

            for each neighbor v of u:           // where v is still in Q.
                alt ← dist[u] + length(u, v)
                if alt < dist[v]:               // A shorter path to v has been found
                    dist[v] ← alt
                    prev[v] ← u
                end if
            end for
        end while

        return dist[], prev[]

    end function
```

Software Implementation

The user space C program will perform several functions. After the maze is constructed, the software will generate a graph by translating the branching points in the maze and path lengths between these points to nodes and internode distances. The graph data will be sent to the FPGA. Once the FPGA completes Dijkstra's algorithm to calculate the optimal path, the software will receive this data and map it to the maze. The program will generate a ternary matrix representation of the maze, containing position information for the walls, paths, as well as the optimal path as calculated by the FPGA, and send that information to the FPGA so that it may be displayed.

Hardware Implementation

The FPGA will receive from the user space C program the graph data and store it into memory. It will, then, perform the appropriate operations, as stated in the pseudocode, getting one node, doing the path comparisons in parallel and storing the temporary values in memory. After it does the whole process to all the nodes it's going to send the optimal path back to software. The block diagram will look something like:



Milestones

1) Milestone 1

a) Construct the maze
b) Generate a graph
c) Implement Dijkstra's algorithm in C
d) Verify the software implementation of the algorithm

2) Milestone 2

a) Implement the algorithm in hardware
b) Send graph data from the software to the hardware
c) Write a testbench to verify the hardware implementation

3) Milestone 3

a) Display the maze as well the optimal path
b) Perform debugging on the design
c) Compare performance of the hardware implementation against the software

References

[1] A. Sharma and S. Hauck, “Accelerating FPGA routing using architecture-adaptive A* techniques,” Proc. - 2005 IEEE Int. Conf. F. Program. Technol., vol. 2005, pp. 225–232, 2005.

[2] M. Tommiska, “Dijkstra’s Shortest Path Routing Algorithm in Reconfigurable Hardware,” Most, no. August, pp. 653–657, 2001.

[3] E. Technologies, “Dijkstra’s algorithm implementation on Fpga Card for Telecom Calculations,” vol. 4, no. 2, pp. 110–116, 2013.

[4] http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Algorithm.