Optimization of Dijkstra's Shortest Path Algorithm on FPGA

Team Members

Ricky Tran
rickydtran@ufl.edu

Christopher Lai
chrislai95@ufl.edu

Wyndham Hudson
caelum@ufl.edu

Project Description

Dijkstra's shortest path algorithm is one of the most important algorithms available for generating the exact optimal solutions to a large set of shortest path problems. Dijkstra's is a greedy algorithm that provides a solution to several real world problems, such as traversing road, router, and telephone networks. Optimizing Dijkstra's algorithm would improve performance in the world of navigation and communication.

The algorithm works by visiting neighboring vertices of a graph beginning at a starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices already examined. It then expands outwards from the initial starting point until it reaches its end point. The algorithm revolves around edge relaxation, where the shortest known path between two vertices can be extended by adding the edge coordinates at the end.

Pseudocode:

```
function Dijkstra(Graph, source):
   for each vertex v in Graph:          //Initialization
       dist[v] := infinity              //Initial distance from source to v set to inf
       previous[v] := undefined         //Previous node in optimal path from source
   dist[source] := 0                    //Distance from source to source
   Q := set of all nodes in Graph       //all nodes in graph are unoptimized-thus in Q
   while Q is not empty:                //main loop
       u:= node in Q with smallest dist[]
       remove u from Q
       for each neighbor v of u:        //where v has not yet been removed from Q.
           alt := dist[u] + dist_between(u,v)
           if alt < dist[v]             //Relax (u,v)
              dist[v] := alt
              previous[v] := u
   return previous[]
```

Algorithms used for calculating the shortest path from a source to any other nodes are computationally expensive. Furthermore, Dijkstra's algorithm can be exploited to allow fine-grain parallelism, by breaking up different tasks. This would make utilizing Dijkstra's algorithm an amenable FPGA-based reconfigurable computing application.

Proposed Approach and Preliminary Design

We will be designing and implementing our application from the ground up and taking into account of different techniques and optimizations that we can perform from class and research papers.

Team Member	Expertise	Work Distribution
Ricky Tran	VHDL, C/C++, Data Structures	Hardware/Software Implementation and Data Generation
Christopher Lai	VHDL, C/C++	Hardware Implementation, Integration, and Testing
Wyndham Hudson	VHDL, C/C++	Software Implementation, Integration, and Testing

Preliminary Design:

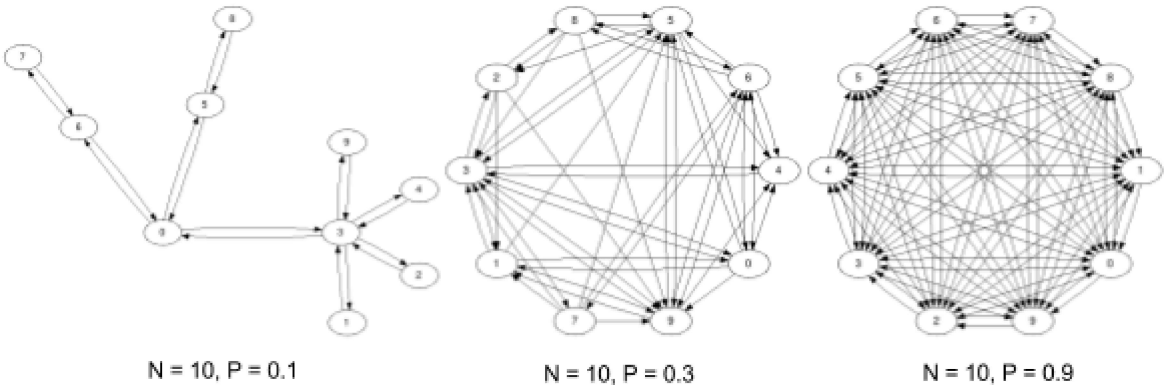
Software Implementation

The C++ userspace program will construct the graph of nodes. The graph data will then be sent to the FPGA. It will wait for the FPGA to finish the computation and store the result from the FPGA in a buffer. Using the same graph it will perform dijkstra's and store the result in another buffer. The results are then compared.

Hardware Implementation

The FPGA will receive the graph data from the C++ userspace and store it in memory. In parallel it will perform the necessary computations for dijkstra's with optimizations and then write back the shortest path to the C++ userspace to then be compared.

Test Data Generation:



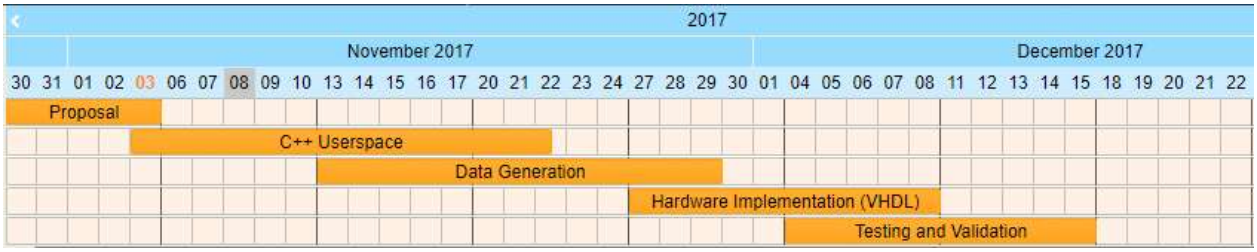
N = 10, P = 0.1 N = 10, P = 0.3 N = 10, P = 0.9

1. We will create a random graph generator that takes in the size (N) of the graph (vertices/nodes) and the connectivity (P) of the graph from 0 to 1 (how sparse is the graph.).
2. The output of this C++ program will be interfaced with the C++ userspace to generate our test data that will be used for both software and hardware implementations.

Demonstration Plan:

1. We will first show how we are generating our test data to be used for our software and hardware implementations. This will take in the two parameters N (size) and P (connectivity) and generate a graph with those constraints with randomly generated weight values.
2. We will be showing a functional and (if time permits) performance comparison of the software and hardware implementations utilizing test test data from the data generation.

Proposed Schedule:



References

T. Grossmann (2017). *Dijkstra Algorithm: Short terms and Pseudocode*. [Online]. Available at: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html [Accessed 3 Nov. 2017].

G. Lei, Y. Dou, R. Li and F. Xia, "An FPGA Implementation for Solving the Large Single-Source-Shortest-Path Problem," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 5, pp. 473-477, May 2016.

K. Sridharan, T. K. Priya and P. R. Kumar, "Hardware architecture for finding shortest paths," *TENCON 2009 - 2009 IEEE Region 10 Conference*, Singapore, 2009, pp. 1-5.

M. Tommiska and J. Skyttä, “Dijkstra’s Shortest Path Routing Algorithm in Reconfigurable Hardware” *Proceedings. 11th Conference on Field-Programmable Logic and Applications*, 2001, pp. 653-657.

A. Sharma and S. Hauck, "Accelerating FPGA routing using architecture-adaptive A* techniques," *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, 2005, pp. 225-232.

“Experimenting with Dijkstra’s algorithm,” 18-Feb-2015. [Online]. Available at: https://gabormakrai.wordpress.com/2015/02/11/experimenting-with-dijkstras-algorithm/ [Accessed: 03-Nov-2017].