EMBS Summer Term Open Assessment 2016 Written Report

Exam Candidate Number: Y1461429

April 28, 2016

1 Problem Analysis

For this open-assessment, the problem was to use an FPGA to solve an NP-hard problem: the travelling salesman problem. The solution must interact with a user via serial to receive a world size and world-id. It will then request and download a world via ethernet, identify the shortest path around the waypoints in this world, avoiding any walls, and finally display and verify the solution.

This problem can be divided into three main sections: interactions with VGA, ethernet and the user; creation of a matrix containing distances between each pair of waypoints; and determining the shortest distance in which every waypoint can be visited.

2 Design

For the design of the solution, two hardware components were used: the Microblaze, which handles interactions with VGA, the user and ethernet, and a custom piece of hardware for solving the travelling salesman problem, and calculating shortest distances using the A* algorithm. The use of special purpose hardware allows many parts of the solution to be pipelined, or run in parallel. As a result, this provides a large increase in speed over a purely software approach for both the A*, and the travelling salesman problem. The remainder of the software was chosen to run on the MicroBlaze as it is not computationally expensive. While running this code on hardware could improve the speed, for example, by providing an entire VGA buffer at once, it is not the limiting factor for the given problem.

3 Implementation

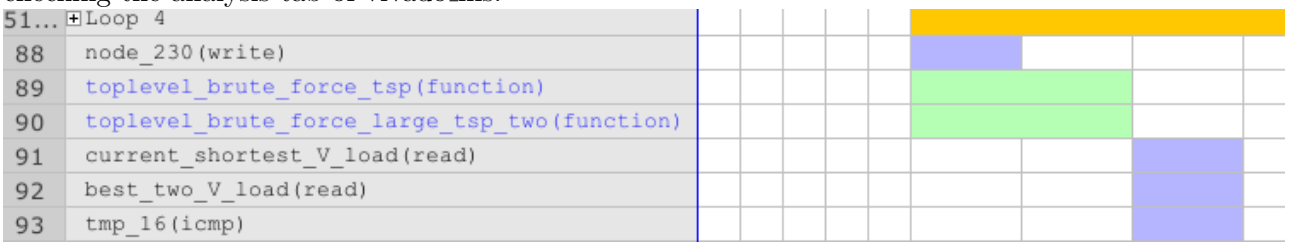
4 Acceleration Strategy

In order to speed up the solution, 3 improvements were made upon the original design.

First, as the original A* algorithm was calculating the cost of each node by counting the number of parents until the start waypoint was reached, a significant speedup was attained by instead keeping track of the cost associated with each coordinate. This comes at the cost of increased memory usage, as each cost must be stored. As there are 3600 coordinates in a large grid, this is a significant amount of additional memory.

Second, while performing the quickperm algorithm, redundant permutations were being generated. This occurs as distances between waypoints are the same in both directions: for example, path A–B–C has the same cost as path C–B–A. As a result, these duplicate results can be removed, significantly reducing the amount of permutations needed. This results in a significant reduction in work: rather than the original 11! permutations needed for a large world, there are therefore only 11!/2. Furthermore, this improves the speed of the brute force approach for all sizes of world. This is implemented in practice by generating 10! permutations rather than 11!, and, for each permutation generated, creating 6 variants by treating the first half of the array as having an extra element.

Finally, the third improvement was made by considering the parallel capabilities of hardware. Unlike in software, it is possible to perform multiple operations at the same time. Therefore, in order to speed up the brute force process, two variations of the quickperm algorithm run in parallel for large worlds. As the size of the world is known, the array being permuted, `a`, and the permutation counting array, `p`, has a known mid-point that can be predetermined. By creating a second copy of the `brute_force_tsp` function, and ensuring the two share no data, the calls to both occur in parallel. This is verified by checking the analysis tab of vivado_hls:



This highlights two iterative permutation algorithms occurring in parallel.

5 Evaluation and Testing

6 Conclusions