GRASP: A Greedy Reconfigurable Approach for Shortest Path

Prasad G. R.
NITK, Surathkal
grprasad_bms@yahoo.com

Dr. K. C. Shet
NITK, Surathkal
kcshet@yahoo.co.uk

Dr. Narasimha B. Bhat
Manipal Dot Net Pvt. Ltd., Manipal
narasim@manipaldotnet.com

Abstract—**This paper presents GRASP ("Greedy Reconfigurable Approach for Shortest Path"), a new shortest path algorithm using reconfigurable logic. It has time complexity O(P), where 'P' is maximum of number of edges along the shortest paths from source to other nodes. It is a modification of Bellman-Ford algorithm and is highly parallel and scalable. Unlike most other shortest path algorithms, GRASP does not need to find the minimum of nodes/adjacent nodes. Hence its FPGA implementation is faster compared to other FPGA implementations. Preliminary experimental results show that a 17-node GRASP runs about 4.7 times faster compared to parallel Bellman-Ford algorithm on Xilinx Virtex II.**

I. INTRODUCTION

Shortest path (SP) problem in graphs is still an active area of research[4], due to the demands for faster SP algorithms by applications like CAD for VLSI[8], robotics[6] and computer networks[3][5]. SP algorithms which run on instruction set based processors like Dijkstra's[1] algorithm and others[4][7], iterate hundreds of instructions and are sequential in nature, and hence have high computation time. Reconfigurable logic based approaches have been used in the past [3][9] to accelerate SP algorithms. But they are slowed down by the process of finding minimum of nodes/adjacent nodes.

Reconfigurable computing[2] achieves high performance by spatially spreading computation on hardware instead of iterating hundreds of instructions on a processor. Reconfigurable computing has execution time close to ASICs with flexibility to reconfigure. It can be used to efficiently and effectively mimic "natural" solutions: an implementation that replicates the way nature tackles analogous problems.

This paper presents GRASP ("Greedy Reconfigurable Approach for Shortest Path"), a new SP algorithm using reconfigurable logic. It has time complexity O(P), where P is the maximum of number of edges along the shortest paths from source to other nodes. It avoids finding minimum of nodes/adjacent nodes and hence is faster compared to other approaches. It is a modification of Bellman-Ford algorithm. In Bellman-Ford algorithm, for every node 'i', distance X_i from source is set as $Min(X_j + D_{ij})$, where X_j is adjacent node's distance from source and D_{ij} is distance between 'i' and 'j'. It is possible to update X_i of all nodes (i =1 to N) in parallel and we call this as parallel Bellman-Ford(PBF) algorithm. In GRASP, a daisy chain is used to to select first value of $X_j + D_{ij}$, which is less than X_i, in a greedy way and thus avoids finding $Min(X_j + D_{ij})$. Use of daisy chain makes it to operate at a much higher clock frequency compared to PBF and hence is much faster compared to PBF. GRASP has more iterations compared to PBF, still it is faster than PBF. GRASP assumes undirected graphs and positive integer edge weights.

The rest of the paper is organized as follows. Section 2 explains related work for finding SP. GRASP and its implementation details are given in Sections 3 and 4 respectively. Section 5 presents experimental results and compares GRASP with other approaches. Section 6 suggests future extensions and the paper concludes with Section 7.

II. RELATED WORK

A. Shortest path algorithms

Dijkstra's algorithm [1] is a popular SP algorithm and is $O(N^2)$. Let x_i be the current distance of node 'i' from source and D be the adjacency matrix. When there is no edge between nodes 'i' and 'j', D_{ij} is set to large value to indicate infinity. Let 's' be the source and 'd' be the destination. Dijkstra's algorithm is as shown in Figure 1.

1. Initialize x_j to D_{sj}, where j =1 to N, and $j \neq s$, set x_s as 0.
   Add 's' to set of labeled nodes F={s}
2. Find minimum among x_j
   $x_i = Min(x_j)$ for j= 1 to N and node 'j' not in F
3. Add node 'i' to F and update each node's distance using
   $x_j = Min(x_j , x_i + D_{ij})$ where j=1 to N
4. Repeat steps 2 and 3 until destination is reached

Figure 1: Dijkstra's Algorithm.

Dijkstra's algorithm has been improved using efficient data structures like radix heap and two level radix heap[7], and have time complexities O(M+NlogC) and O(M+NlogC/loglogC), where C is edge weight, M is number of edges and N is number of nodes. A recent improvement[4] has time complexity $O(M+D_{max}\log(N!))$, where D_{max} is maximal number of edges incident at a vertex. Implementation of Dijkstra's algorithm on reconfigurable logic is presented in [3] and this uses a comparator tree to find minimum instead of using a loop. But the algorithm has to repeat steps 2 and 3(Figure 1) until destination is reached and hence its time complexity is O(N). Ralf Moller[6] has reformulated Dijkstra's algorithm and implemented that using the concept of signal propagation, and has time complexity O(L).

Bellman-Ford algorithm is as shown in Figure 2 and has time complexity $O(N^3)$. Here, distance of a node is updated using the relation $X_i = Min(X_j + D_{ij})$ and this is continued till there are no changes in X_i. The parallel implementation of Bellman-Ford algorithm updates all node values in parallel and has time complexity O(P). For the example graph in Figure 3 trace is as shown in Figures

K. Elleithy (ed.), *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering,* 393–398.
© Springer Science+Business Media B.V. 2008

4 to 7. Nodes which are at infinity are not shown in Figures.

1. Initialize x_j to D_{sj}, where j =1 to N, and j ≠ s,
   set x_s as 0.
2. Update each node position by the relation
   x_i=Min(x_j + D_{ij}) for j = 1 to N and j ≠ i
3. Repeat step 2 till there are no changes.

Figure 2: Bellman-Ford Algorithm.

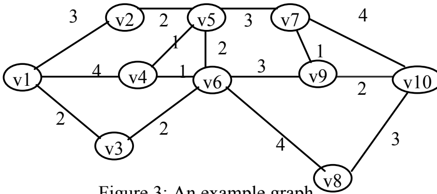

Figure 3: An example graph.

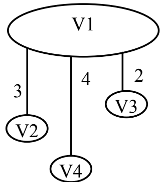

Figure 4: After first clock cycle.

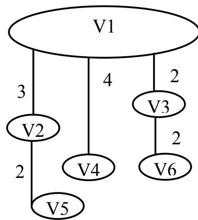

Figure 5: After second clock cycle.

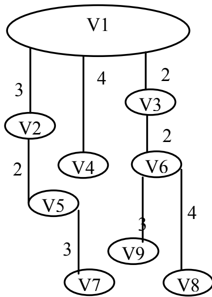

Figure 6: After third clock cycle.

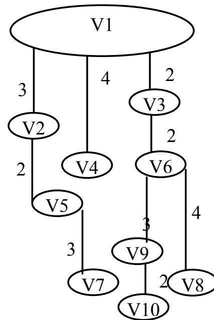

Figure 7: At the end.

B. *Problems and opportunities*

Most of the existing algorithms find minimum of nodes/adjacent nodes and are sequential in nature (select nodes one by one), and hence have high computation time.

Finding minimum can be avoided by using a daisy chain to select a better value for a node. This approach is faster compared to the approach which finds minimum. In Parallel Bellman-Ford(PBF) algorithm, nodes that are having same number of edges along the shortest path from source get fixed in parallel overcoming the sequential selection of nodes. In GRASP also nodes get fixed in parallel fashion.

III. GRASP

In GRASP, N nodes synchronously update their X_i using a daisy chain. Each node 'i', finds $X_j + D_{ij}$ for all adjacent nodes 'j', in parallel. It scans adjacent nodes from 1 to N to find a X_j+D_{ij} that is less than its current X_i using daisy chain and the first value thus found is set as new X_i and corresponding 'j' is set as previous node of 'i'. We associate with each node a flag O_i and is set to 0 in the beginning of the clock cycle, and is later set to 1 whenever there is a change in X_i in that clock cycle. If 'i' does not find any $X_j + D_{ij}$ as less than current X_i, then old position is retained and O_i remains as 0. This process of updating X_i needs adjacent node distances X_j and D_{ij}'s from adjacency matrix. Given this information, a node can update its value independently and this makes GRASP scalable. The algorithm stops when there are no changes in any X_i or, in other words when all O_i are 0.

For graph in Figure 3, let v1 be the source and v10 be the destination. The trace of GRASP is as shown in Figures 8 to 13 and the algorithm is as shown in Figure 14. Initially X_i of v1 is 0 and all others will have 99999(indicating infinity and nodes which are at infinity are not shown in Figures). At first clock cycle, v2 finds X_1 + D_{21} (i.e. 0+3) as less than X_2 (i.e.99999) and hence sets X_2 to 3. Similarly v3 and v4 set their X_i to 2 and 4 respectively. In next clock cycle, v5 and v6 set their X_i to 5 and 4 respectively. In third clock cycle v7, v8 and v9 set their X_i to 8, 8 and 7 respectively. In fourth clock cycle, v10 sets its X_i to 12 through v7, as through v7 it finds X_i as 12 lesser than 99999. At this time though through v9 it is possible to set v10's X_i to 9, it selects v7, as it is seen first. In next clock it sets X_i to 11 through v8 and at the end it sets X_i to 9 through v9 and stops.

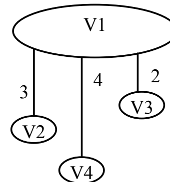

Figure 8: After first clock cycle.

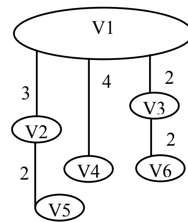

Figure 9: After second clock cycle.

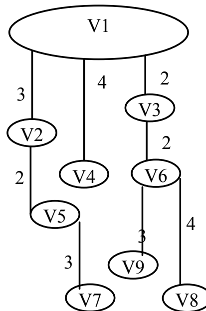

Figure 10: After third clock cycle.

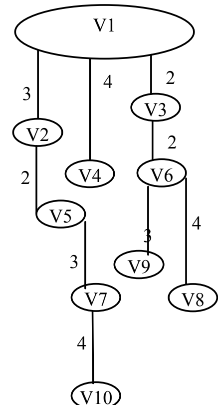

Figure 11: After fourth clock cycle.