NAMST-A: New Algorithm for Minimum Spanning Tree (Adaptive) using Reconfigurable Logic

Prasad G. R . NITK, Surathkal, India

K. C. Shet NITK, Surathkal, India

Narasimha B. Bhat Manipal Dot Net Pvt Ltd Manipal, India

Summary

This paper presents NAMST-A (New Algorithm for Minimum Spanning Tree- Adaptive), a new reconfigurable logic based algorithm for finding minimum spanning tree (MST). It is an extension of NAMST [7]. It uses ball and string model, and has a time complexity O(N), where 'N' is the number of nodes. It is faster compared to other existing MST algorithms as it does not need to find the minimum of nodes/adjacent nodes. On Xilinx Virtex II Pro kit, NAMST-A takes 428.92 ns for finding MST of size 28 in a graph of 17 nodes.

Key words:
Reconfigurable computing, minimum spanning tree, ball and string model, FPGA.

1. Introduction

Finding minimum spanning tree is still an active area of research [1] [3] [4] [9] [10], due to the demand for faster algorithms by the applications that use it, like CAD for VLSI, wireless communication, distributed networks etc. Most of the existing MST algorithms iterate hundreds of instructions and hence have high execution time.

Reconfigurable computing [2] achieves high performance by spatially spreading computation on the hardware instead of iterating hundreds of instructions on a processor. Reconfigurable computing has execution time close to that of 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 NAMST-A ("New Algorithm for Minimum Spanning Tree-Adaptive"), a new MST algorithm using reconfigurable logic, and is an extension of NAMST [7], to accelerate the algorithm for graphs of large edge weights. NAMST and NASMT-A have time complexity O(N), where 'N' is the number of nodes. They do not need to find minimum of nodes/adjacent nodes, like other existing MST algorithms do. Hence they are faster compared to the other MST algorithms.

Manuscript received May 5, 2008
Manuscript revised May 20, 2008

NAMST is an extension of NATR (New Algorithm for Tracing Routes) [6], our earlier work on finding shortest paths in large graphs. Finding shortest path mimics the formation of ball and string model [5] [6]. In finding the shortest path, all nodes except the source (which is fixed) fall down synchronously from the source by comparing their positions with that of the adjacent nodes, and stop on reaching shortest distance from the source. A stopped node's position is the shortest distance between that node and the source. In NAMST too, all nodes except the source, fall down synchronously from the source. When a string between a fixed node 'j' and a free node 'i' is stretched to full, then the free node 'i' is fixed and 'j' is set as the previous node of 'i'. Also the positions of all free nodes are set 0, and the free nodes are made to fall gain.. This is continued till all nodes get fixed, and this yields the MST. Given its adjacent node's information, a node will fall independently and this makes NAMST scalable. NAMST assumes undirected graphs with positive integer edge weights.

NAMST-A is similar to NAMST, with the only difference being that the nodes fall by multiple steps in the former instead of single unit steps in the latter. This is done to accelerate the algorithm for graphs of large edge weights.

The rest of the paper is organized as follows. Section 2 explains the ball and string model and its formation. NAMST-A and its implementation details are given in Sections 3 and 4 respectively. Section 5 presents the results and Section 6 explains the scalability of the algorithm. The paper concludes with Section 7.

2. Ball and String Model (BSM)

Ball and string model (BSM) of a graph is a network of balls connected by strings, where balls represent the nodes and strings represent the edges of the graph. For the graph shown in Figure 1, its equivalent BSM is as shown in Figure 2. In this figure, a straight line represents a fully stretched string, and a curved line represents a string with slack.

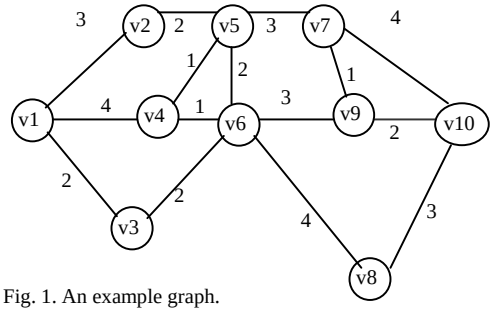

Fig. 1. An example graph.

Figures 3 to 6 illustrate the formation of BSM. In these Figures, nodes are denoted by circles, strings by lines and the fixed nodes by hatching. V1 is the source, which is fixed. Initially (Figure 3) all nodes (balls) are at a distance of 0 from the source. Keeping V1 fixed, the other nodes are released, and their motion is shown in Figures 4 to 6.

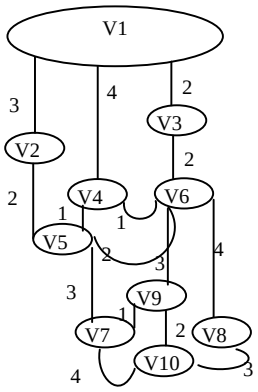

Fig. 2. Ball and string model for the graph of Fig. 1.

Figure 4 shows the positions of the nodes after they fall down by a unit distance. At this point of time no string is stretched to full. Figure 5 shows the positions of the nodes when they fall by a distance of 2 units, and now the string between the nodes v1 and v3 is stretched to full. Hence v3 cannot fall down further and gets fixed at 2. Figure 6 shows the positions of the balls after they fall by a distance of 3 units, and now v2 gets fixed. This continues and finally we obatin the BSM as shown in Figure 2.

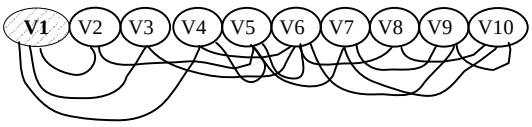

Fig. 3. Initially all balls are together and v1 is fixed.

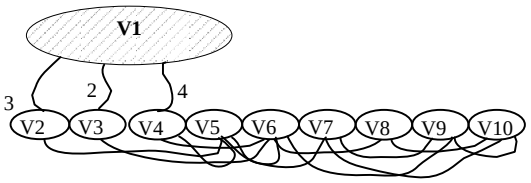

Fig. 4. Balls after falling by a distance of 1.

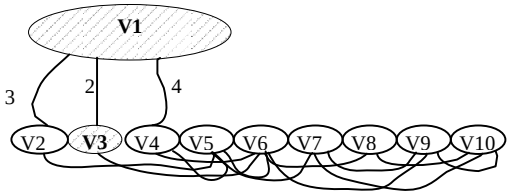

Fig. 5. Balls after falling by a distance of 2, v3 is fixed.

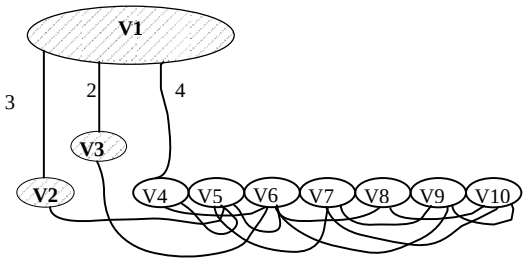

Fig. 6. Balls after falling by a distance of 3, v2 is fixed.

3. NAMST-A

In the formation of BSM, whenever a node gets fixed, if all the free nodes are brought to 0 and are made to fall again, it results in the formation of MST, as at each time a free node close to the fixed node/s is selected and fixed.

For the graph in Figure 1 with v1 as source, the trace is as shown in Figures 7 to 12, and at the end we get the MST as in Figure 13. Figure 7 shows the positions of the nodes when they are together at 0, with v1 as fixed node. Figure 8 shows the positions of the nodes after they fall down by a unit distance and, at this point of time no string is stretched to full. Figure 9 shows the positions of the nodes when they fall by a distance of 2 units and now the string between the nodes v1 and v3 is stretched to full. Hence v3 is fixed at 2 and at the same time, the positions of all free nodes are set to 0 as shown in Figure 10. The free nodes are made to fall again and, after falling by a distance of 2, node v6 is fixed with v3 as its previous node. This continues till all nodes get fixed and at the end we get MST as shown in Figure 13.

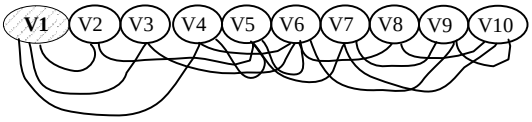

Fig. 7. Initially all balls are together and v1 is fixed.

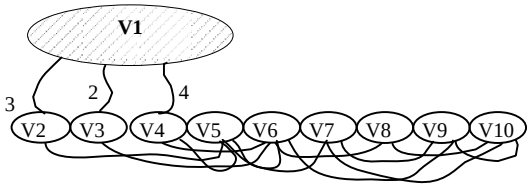

Fig. 8. Balls after moving by distance of 1.

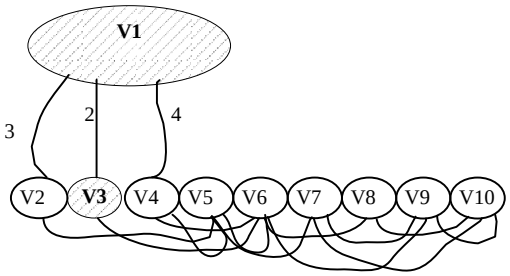

Fig. 9. Balls after moving by distance of 2, v3 is fixed.

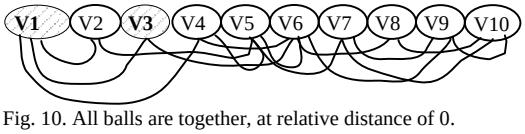

Fig. 10. All balls are together, at relative distance of 0.

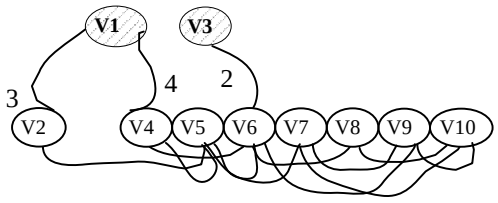

Fig. 11. Balls after moving by distance of 1.

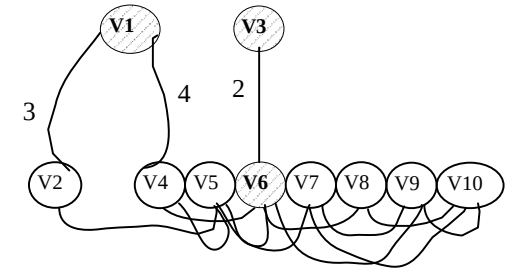

Fig. 12. Balls after moving by distance of 2.

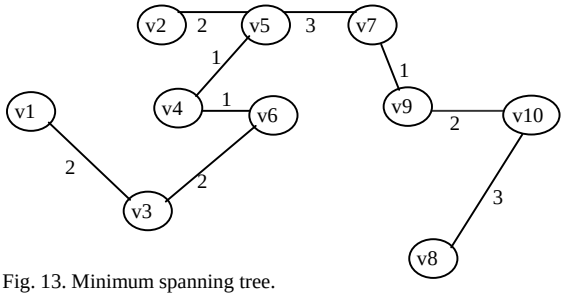

Fig. 13. Minimum spanning tree.

When the nodes are falling, more than one node may be eligible to get fixed. This happens when multiple strings are fully stretched simultaneously. But only one node is selected at a time, to avoid non optimal solution, and hence the time complexity is O(N). Consider the situation in Figure 14. In this when the free nodes (unconnected nodes) fall down by a distance of 3 units, nodes v7 and v9 satisfy the condition to get fixed. But fixing both nodes results in a non optimal solution (to connect v7 and v9 to MST, edge weights taken are 3+3=6). Instead, either node v7 or v9 is fixed first and, then the other node is fixed using the link between them (to connect v7 and v9 to MST, edge weights taken are 3+1=4). This gives the optimal solution.

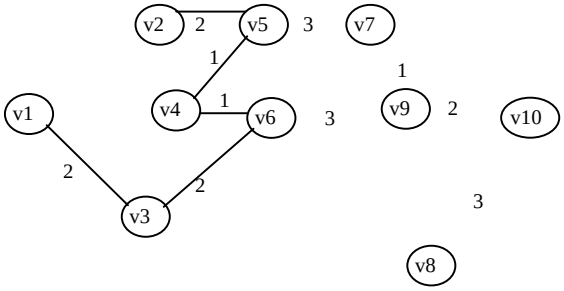

Fig. 14. Minimum spanning tree, during construction.

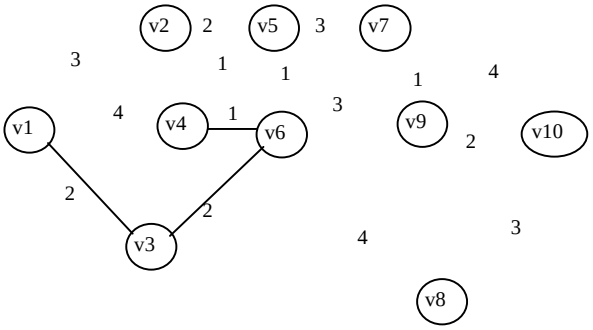

Fig. 15. Minimum spanning tree, during construction.

Whenever a node gets fixed, there may be more than one adjacent node satisfying the condition (a string stretched to the full) to become previous node. In this situation too, only one is selected using daisy chain. To give an example, in Figure 15, when the node v5 falls by a distance of 1 unit,

nodes v4 and v6 satisfy the condition to become the previous node of v5. In NAMST-A, v4 is selected as the previous node of v5. The NAMST-A algorithm is as shown in Figure 16.

If the nodes fall by a unit distance (K=1) in each step, then 'S' steps are needed to build the MST, where 'S' is the size of the MST. This is the *basic version of NAMST*, and is as illustrated in Figures 7 to 13. For graphs of large edge weights, the algorithm can be accelerated by making the nodes to fall by more than a unit distance. This is the adaptive version of NAMST. The algorithm adapts to the string lengths (edge weights) by means of a variable step size SZ. SZ is the number of units by which the nodes fall in a step. Initially SZ is set to 1. Later, if a 'move' is successful for all nodes, then the SZ is multiplied by 'K' else the SZ is divided by 'K', where 'K' is the acceleration factor. A successful 'move' is one in which the actual separation between the nodes does not exceed the string length between nodes; which is the value in the adjacency matrix D. In case of a failure the separation exceeds the value in adjacency matrix, and logically it means that the string between nodes is broken. The adaptive version is illustrated in Figures 17 to 23 for K=2.

To show how the adaptive version reduces the number of steps required to find the MST, consider the graph shown in Figure 17. The trace of the adaptive version (with K=2) for this graph is as shown in Figures 18 to 23. Initially SZ is 1 and all nodes are together at 0 as shown in Figure 18. In the first step all nodes fall down by a unit distance as shown in Figure 19 and move is successful for all nodes. Hence SZ is doubled to 2. The nodes are again successful in falling by 2 units and again SZ is doubled to 4. Once again the nodes are successful in falling by 4 units and SZ is doubled to 8. But in the next step the string between the nodes v1 and v3 breaks, when v3 tries to move by 8 units (from 7 to 15), as the maximum separation allowed between the nodes v1 and v3 is 10 (value from D). The actual separation between v3 and v1 is 15-0=15. Hence move is a failure for v3 and all nodes remain at the old positions with SZ being halved to 4. In next step, again move is a failure as new position of v3 is 11 and the actual separation (11-0=11) is greater than 10. Hence SZ is halved to 2 and in the next step move is successful. In the 9^{th} step, the string between the nodes v1 and v3 is stretched to full. Hence v3 is fixed at 10. This continues till all nodes get fixed. Though the MST size is 160, it is found in 99 steps. Thus for large edge weights adaptive version reduces number of steps required to find MST.

1. For all nodes, position X_i is set to 0. For all non source nodes set status flag O to 0 and for source set this to 1. PV is previous node, TO and TPV are temporary status flag and previous nodes, set step size SZ to 1. MF stands for move failure and on failure it is set to 1 else it will be 0.

2. While not all nodes are fixed loop

```
for all i=1 to N do in parallel
   X_i = X_i + SZ
   for j=1 to N do in parallel
      if ((X_i > D_ij ) then
         MF_i =1
      else
         if (X_i = D_ij and O_j =1) then
            TO_i = 1
            TPV_i =j
         end if
      end if
   end for loop
end for loop

MF=0
for i = 1 to N do
   MF=MF or MF_i
end for loop

if(MF=1) then
   if(SZ > =K) then
      SZ=SZ/K
   else
      SZ=1
   end if
else
   sflag=1
   for i=1 to N loop
      if(TO_i =1 and sflag=1)then
         set O_i to 1
         set PV_i to TPV_i
         sflag=0
      end if
   end for loop
   SZ=SZ*K
end if

if none of the nodes are fixed in the current step
      update incremented positions as X_i
else
      set all node positions X_i to 0
end if

end while loop
```

Fig. 16. NAMST algorithm.

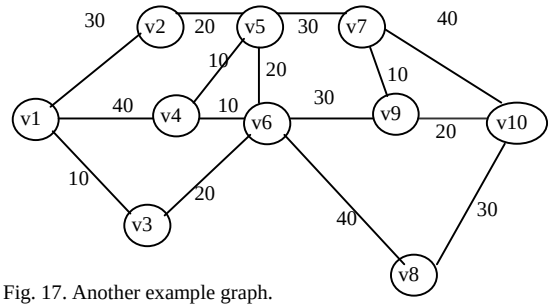

Fig. 17. Another example graph.

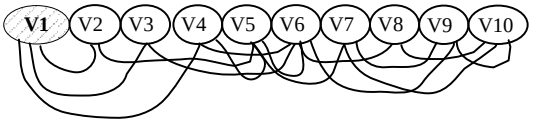

Fig. 18. Initially all balls are together and v1 is fixed and step_size is 1.

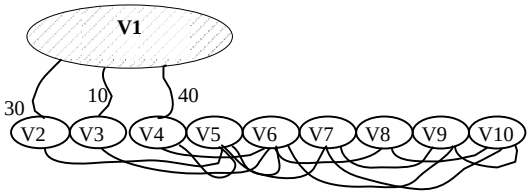

Fig.19. Balls are successful in moving by distance of 1 and step_size is doubled to 2.

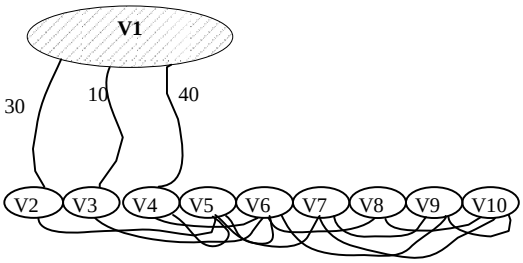

Fig.20. Again balls are successful in moving by a distance of 2 and step_size is doubled to 4.

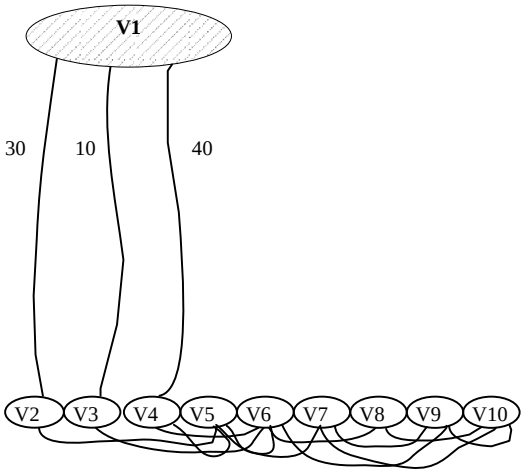

Fig.21. Again balls are successful in moving by a distance of 4 and step_size is doubled 8.

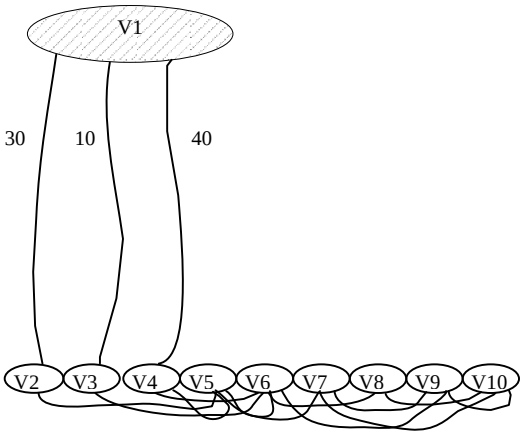

Fig.22. v3 fails to move by 8, and hence step_size is halved to 4 and old positions are retained.

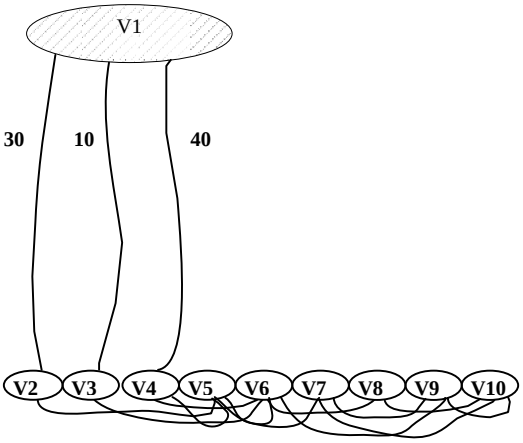

Fig.23. v3 fails to move by even 4, and hence step_size is again halved to 2 and old positions are retained.

4. Implementation

NAMST-A is implemented using Xilinx Virtex II Pro development kit [11] with XC2VP30 FPGA and is coded in VHDL. ChipScope pro 8.2i is used to check the results.

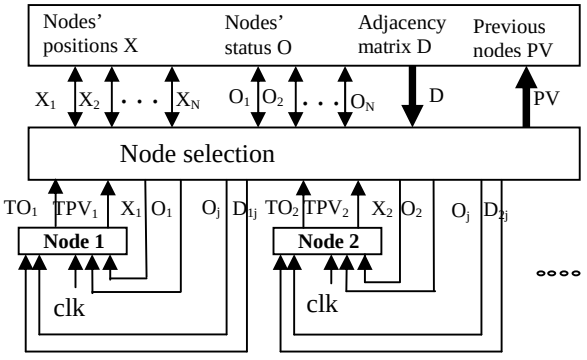

Fig. 24. NAMST-A, as a collection of nodes.

As shown in Figure 24, NAMST-A/NAMST consists of 'N' nodes falling synchronously with the clock. To fall down, a node takes information of the adjacent nodes i.e. adjacent node position and status flag (fixed or moveable). Node position(X) and previous nodes (PV) are declared as signals (integer array) and status flags (O) is declared as a signal of type std_logic_vector of size N. Adjacency matrix D is declared as two dimensional constant array. A signal by name "load" is used to initialize values (when load=0) and after the initialization, load is made 1. In subsequent clocks (when load=1), nodes move as described in the algorithm, and the algorithm comes to a halt when all the nodes get fixed.

4.1 Node structure of Basic version/NAMST

Node structure for the basic version is as shown in Figure 25. A node increments its position by 1(step_size) and compares that with the adjacency matrix value D_{ij} to set flag E_j. If $X_i=D_{ij}$ then E_j is set to 1, otherwise it is set to 0. If any E_j is 1 with the corresponding O_j as 1, it implies that the string between node 'i' and its adjacent node 'j' is stretched to full, and hence 'i' is to be fixed. So TO_i is set to 1 and TPV_i is set to j, where TO_i and TPV_i are temporary status flag and temporary previous nodes of node 'i' respectively. More than one adjacent node may satisfy condition to become previous node of 'i' but only one node (low numbered) is selected using a daisy chain. As said earlier only one node is to be fixed at a time to get optimal solution and for this purpose TO and TPV are used. Using TO and TPV, and a daisy chain, one node (low numbered) among the many eligible nodes is fixed, as shown in Figure 26. In Figure 26, part A shows selection of a node and setting its status flag, part B shows setting previous node and part C shows setting X for the next clock cycle. In the current clock cycle, if any node gets fixed then positions of free nodes are set to 0 else positions are incremented by 1.

4.2 Node structure of NAMST-A

For the adaptive version, the node structure is same as that of the basic version, except that an integer signal step_size, SZ is used in place of constant 1, at the adder input. The comparator will have output C_j which tells whether $X_i > D_{ij}$ (if so move is a failure). In a clock cycle, if a move is a failure for any node, then the step_size SZ is divided by 'K' (if step_size > K else step_size is set to 1). On a successful move step_size SZ is multiplied by 'K'.

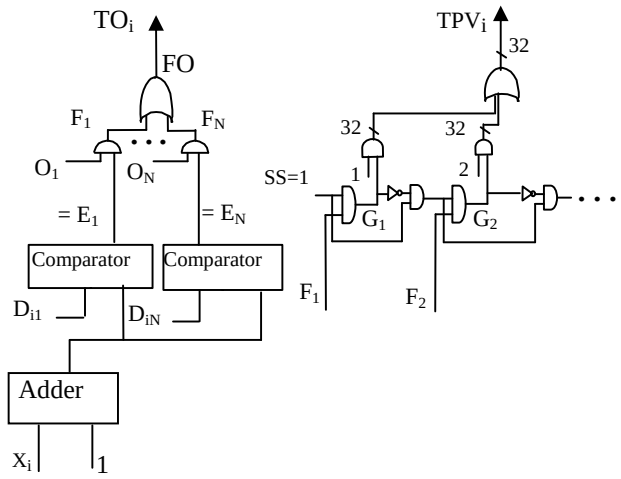

Fig. 25. Node structure.

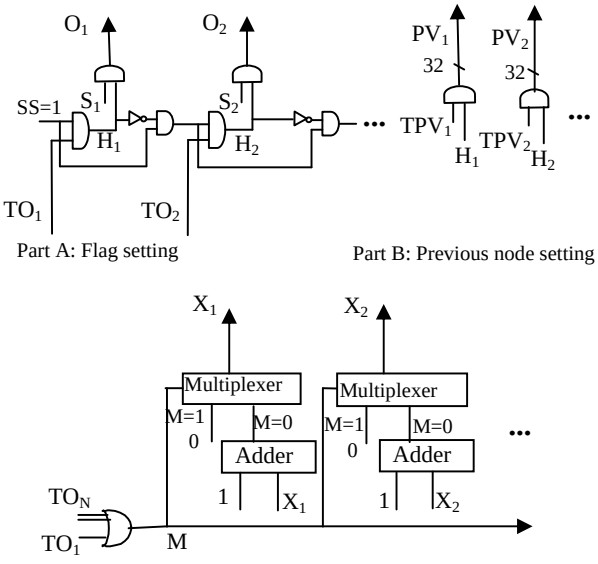

Fig. 26. Node selection through daisy chain.

4.3 Resource and memory requirement

With each node consisting of N comparators, resource needed is $O(N^2)$ and because of storing D in memory, storage required is $O(N^2)$. But with Z (Z<<N) adjacent nodes, we need to allocate resources and memory for Z adjacent nodes only, i.e. Z*N comparators and 2*Z*N memory for adjacent node information (2*Z=Z for edge weights + Z for adjacent node numbers). Hence with Z

adjacent nodes, the resource required is O(N), and the memory required is also O(N).
Table 1 Results of NAMST-A.

	NAMST (Basic)			NAMST-A (Adaptive)		
N	**5**	**10**	**17**	**5**	**10**	**17**
Max. Operating Clock frequency	122.84	109.38	103.00	105.26	97.42	81.60
LUT %	1	3	7	2	6	12
Gate Count	4,061	7,880	14,058	8,606	18,329	33,324
Clock cycles	8	17	28	10	20	35
Execution time in ns using Max. operating clock(FPGA)	65.12	155.42	271.84	95.00	205.29	428.92
Actual Clock used MHz (Virtex- II Pro)	100	100	100	100	50	50
Execution time in ns (Virtex II - Pro)	80	170	280	100	400	700
Execution time in ns (Pentium IV)	4000	40,000	1,75,000	4000	30,000	1,30,000

5. Results

The experiment is conducted for graphs of sizes 5, 10 and 17 nodes by placing all nodes in FPGA. Results are given in Table 1. NAMST-A is also implemented in C on Pentium IV processor running at 2.66GHz and having 1GB RAM. The results clearly show that FPGA implementation is good. A graph is also plotted for execution time as shown in Figure 27.

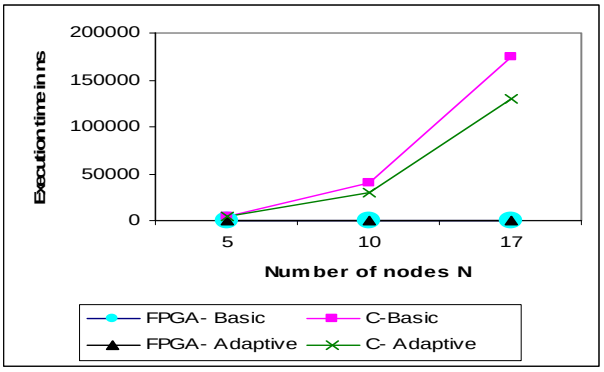

Fig. 27. Execution times for NAMST.

6. Scalability

NAMST-A handles graphs that are larger than an FPGA's capacity. This section explains how computation to be done in a clock cycle is divided into smaller steps, and how these steps are sequentially executed on FPGA.

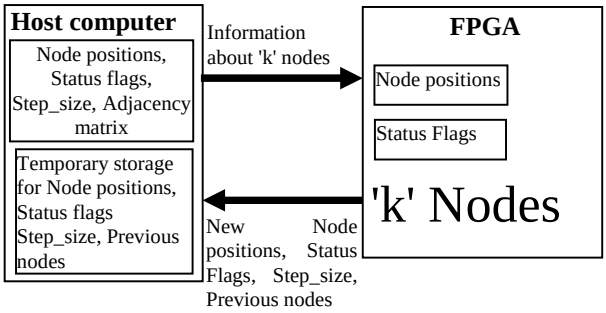

Fig. 28. Interface between FPGA and host computer for 'k' nodes computation.

Figure 28 shows the logical interface between the host computer and the FPGA. If the FPGA can accommodate only 'k' nodes (1<=k<=N), then computation to be done in an iteration (clock cycle) is divided in to N/k steps and in each step, computation of 'k' nodes is done. To start computation for an iteration, X and O of all nodes are loaded into FPGA memory, since a node's computation needs adjacent node information. First 'k' node's X_i and O_i, and first 'k' rows of D are sent to the FPGA and computation is done for first 'k' nodes. Node structure is same for all nodes except current node's X, O and D, and hence these values are loaded when computation for a node has to be done. After the computation, results from the FPGA are stored in a temporary space in the host computer for use in the next iteration. Similarly computation of all nodes is done by taking 'k' nodes at a time and once computation for the current iteration is over, the values held in temporary space are updated as new values. Before starting computation for the next iteration, once again X and O are loaded in to FPGA.

Suppose we want to find MST for a graph of 10 nodes using an FPGA that can accommodate only 4 nodes, in that case computation to be done in an iteration is divided in to three steps as shown in Figure 29.

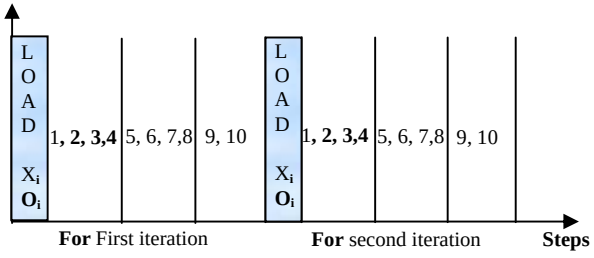

Fig. 29. Computation of '4' nodes in a clock cycle.

7. Conclusions

A new reconfigurable logic based minimum spanning tree algorithm (NAMST-A) is proposed. NAMST-A has a time complexity O(N). The algorithm is based on ball and string model. Unlike most other existing algorithms, NAMST-A does not need to find minimum of nodes/adjacent nodes, which makes it faster compared to other MST algorithms. In addition, it is implemented on reconfigurable logic to have high performance close to that of ASICs. In the basic version/NAMST, nodes always fall down by a unit distance and hence 'S' clock cycles are needed to find MST, where 'S' is the size of the MST. The adaptive version of NAMST (NAMST-A) takes less than S clock cycles to find MST in graphs with large edge weights(>5), but takes more than 'S' clock cycles in case of graphs with small edge weights. Hence for graphs with small edge weights basic version is efficient and for graphs with large edge weights adaptive version is efficient.

References

[1] Aili Han and Daming Zhu, *"DNA Computing Model for the Minimum Spanning Tree Problem"*, Proceedings of the eighth international symposium on symbolic and numeric algorithms for scientific computing (SYNASC'06), 2006, pp 372-377.

[2] Andre DeHon and John Wawrzynek, *"Reconfigurable Computing: What, Why, and Implications for Design Automation"*, Proceedings of 1999 Design Automation Conference, June 1999, pp 610-615.

[3] David A Bader, Guojing Cong, *"A fast, parallel spanning tree algorithm for symmetric multiprocessors (SMPs) "*, Journal of parallel and distributed computing, 2005, pp 994 – 1006.

[4] David A. Bader, Guojing Cong, *"Fast Shared-Memory Algorithms for Computing the Minimum Spanning Forest of Sparse Graphs"*, Journal of Parallel and Distributed Computing Volume 66, Issue 11, Nov 2006, pp 1366 – 1378.

[5] Paolo Narvaez, Kai-Yeung Siu and Hong-Yi Tzeng, *"New Dynamic SPT Algorithm Based on a Ball and String Model"*, IEEE transactions on Networking, Dec 2001, pp 706-718.

[6] Prasad G. R., K. C. Shet and Narasimha B. Bhat, *"NATR: A New Algorithm for Tracing Routes"*, presented in International Joint Conferences on Computer, Information, and System Sciences, and Engineering, Dec 3-12, 2007.

[7] Prasad G. R., K. C. Shet and Narasimha B. Bhat, *"NAMST: A New Algorithm for Minimum Spanning Tree using Reconfigurable Logic"*, presented in International Conference on Information Systems and Technology, Dec 14-15, 2007, Thrissur, Kerala, India, pp 37-42.

[8] Sabih H. Gerez, *"Algorithms for VLSI Design Automation"*, John Wiley & Sons (Asia) Pte. Ltd., 2004.

[9] Lixia Hanr and Yuping Wang, *"A Novel Genetic Algorithm for Degree-Constrained Minimum Spanning Tree Problem"*, International Journal of Computer Science and Network Security, VOL.6 No.7A, July 2006, pp 50-57.

[10] Seth Pettie and Vijaya Ramchandran, "An optimal minimum spanning tree algorithm", ACM journal, VOL 49, No 1, Jan 2002, pp 16-34.

[11] www.xilinx.com, XUPV2P user guide.

Prasad G R is a research scholar at National Institute of Technology, Karnataka, Surathkal, INDIA. He received his M.Tech degree in Computer Science & Engineering from Bangalore University in 1999 and B.E Degree in Computer Science & Engineering from Bangalore University in 1995. His research interests include Reconfigurable computing.

Dr. K.C Shet is a professor in Dept. of Computer Engineering, National Institute of Technology, Karnataka, Surathkal, INDIA. He has more than 36 years of experience in teaching and research. He holds a Ph. D. from IIT Bombay, INDIA. He is a member of Computer Society of India, and Indian Society of Technical Education. He is a Fellow of Institution of Engineers (INDIA). His research interests includes software testing, Security Solution for Web Services, Cyber Laws, Anti spam solutions, Wireless Networks, Mobile Computing, Ad hoc Networks. He has published more than 200 papers in refereed conference proceedings and journals.

Dr. Narasimha B. Bhat is CEO and Founder of Manipal Dot Net (www.manipal.net), a technology startup in Manipal, Southern India. He was R&D Director at Synopsys USA. He has a PhD from UC Berkeley, ME from IISc Bangalore and BE from MIT, Manipal. His research interests include Reconfigurable Computing, Embedded Systems and EDA.