PRL Project 3 Documentation – Parallel preorder

Author: Martin Krajňák
Login: xkrajn02
April 25th 2018

Algorithm analysis

The goal of the algorithm is to take an input string, convert it to the binary tree and do the preorder traversal. Each edge is converted to two edges – a forwarding one and reverse one. Each processor is assigned one edge of the tree and perform the following steps:

1. Calculate the successor edge via Euler tour (except the last processor – value set to -1)
2. Calculate the predecessor edge (except the first processor – value set to -1)
3. Set rank (different from processor id) to 1 (except the last processor – value set to 0)
4. Perform the List ranking algorithm – calculate distance to the end of the linked edges
5. Reverse the result to the position from the start
6. Mark the forwarding edges and calculate vertices they are leading into
7. Send the values to the first processor – each value represent the index of the input string character which is printed out (except the very first character since the root of the tree is always the starting point), non forwarding edges, in my case even edges, are skipped

Time and costs analysis

Time complexity of the tasks mentioned above:

1. Successor calculation	$O(n)=O(1)$
2. Predecessor calculation	$O(n)=O(1)$
3. Rank initialization	$O(n)=O(1)$
4. List ranking	$O(n)=O(\log_2 n)$
5. Reversing the result	$O(n)=O(1)$
6. Marking forward edges and calculating vertices	$O(n)=O(1)$

Overall time complexity	$t(n)=O(\log_2 n)$
Overal cost	$c(n)=O(n*\log_2 n)$

Implementation details

The important part of the program execution is done by the script test.sh as it takes the input string and calculates the number of required processors for computation. The number is calculated by formula *2*n-2* where *n* is length of the string and is passed to *mpirun* command as an argument. String transformation to the binaryu tree can be seen on Illustration 1. Every processor calculates his edge by adding one to his rank (id).

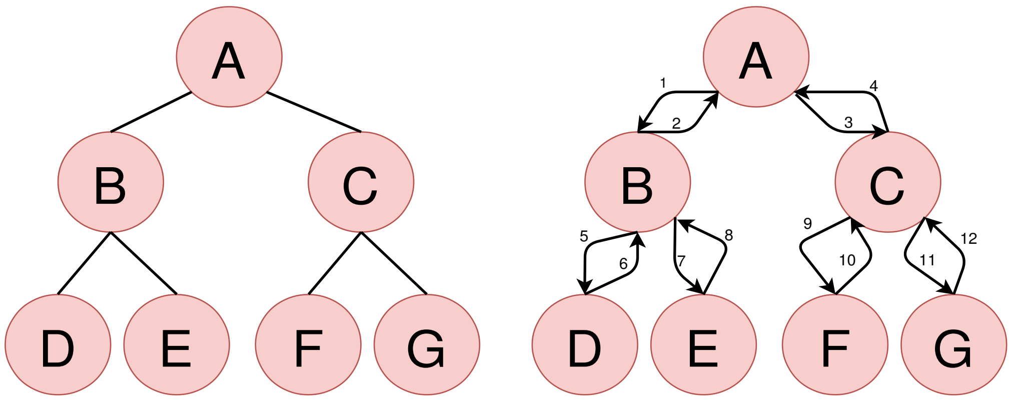

Illustration 1: Conversion of the graph edges

Euler tour

Algorithm requires that every edge knows his successor which can be calculated via Euler tour[1] in constant time. The proces of deciding the successor depends on dividing edges to categories. Odd edges are always forwarding edges, reverse edges are even. Additionally, the edge number modulo 4 give us the exact type of the edge. For example edge 3 leads to the vertex 3 or C, so his successor is the edge leading to left son on vertex C - number 9 and so on. After calculating successors each processor send a message to his successor with his id, so each processor knows also his predecessor.

List ranking

This part calculates the distance to the end of the Euler path for each edge. In the beginning of the process each processor (vertex) know his edge, successors edge and predecessors edge, so we can imagine processors like one linked array list of edges. Furthermore, each one of them has been assigned with *rank* value (1 for all, 0 for last) which will be changing throughtout the iterations. Algorithm divides procesors to three groups, the first one has no predecessor, the last one has no successor and the middle ones that have both. The execution process is described in the part of the Illustration 2 and takes *n*log* n iterations. The results are reverted after the execution ends to obtain distance from the beginning.

Testing and conclusion

Testing took place on server Merlin, measurements were done by functions from *chrono* library to measure only execution time of the algorithm. The goal of testing was to confirm the theoretical time complexity. The measured values are displayed in Chart 1 and they are approximately confirming the calculated theoretical complexity. Small anomalies might be caused by the fact that the server is being used constantly by several users and load of the server depends on their tasks. The testing was stopped on the string ABCDIJKLMN, longer strings require more procesors that are present on machine.

1 https://en.wikipedia.org/wiki/Euler_tour_technique

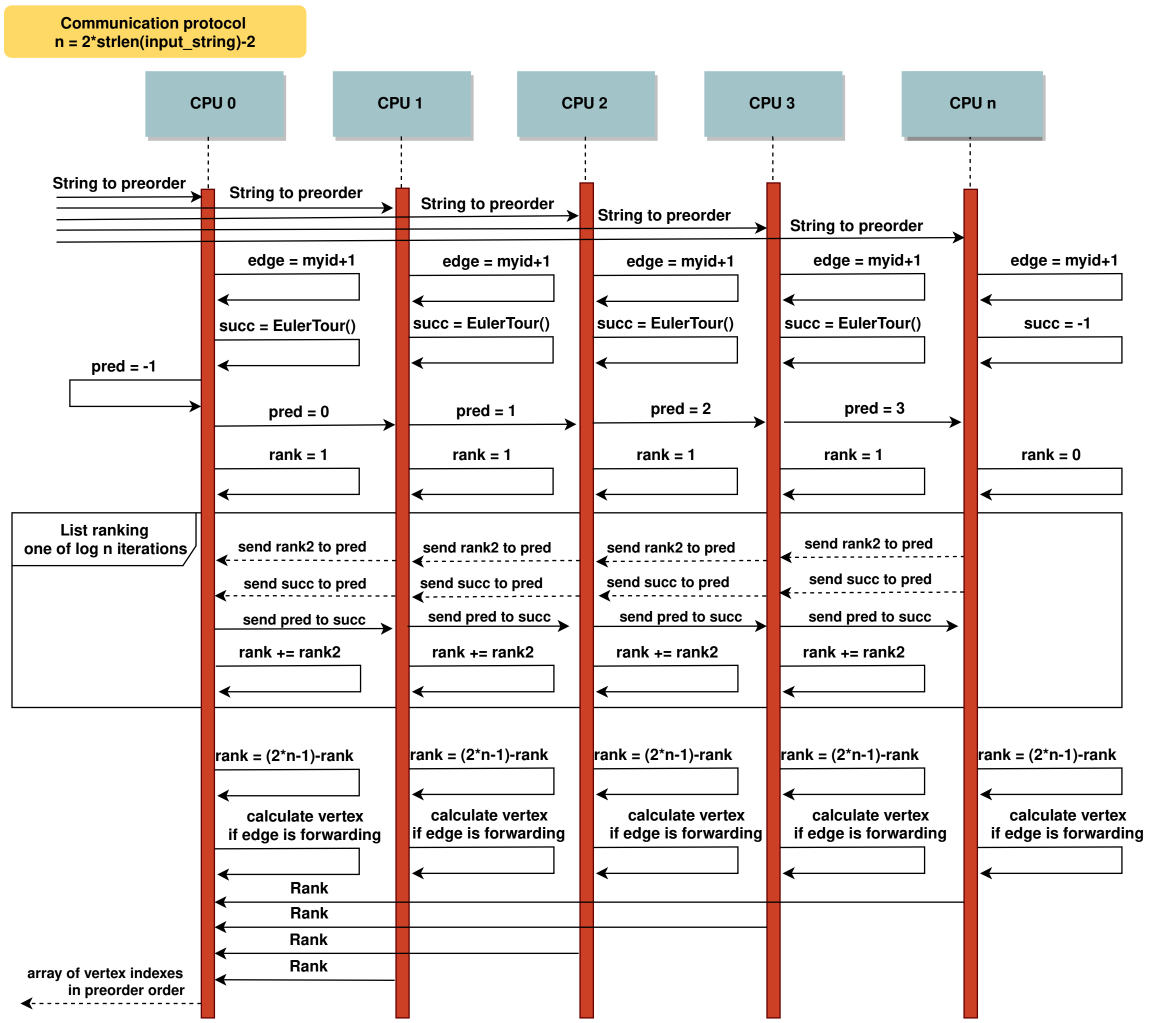

Illustration 2: Communication between procesors

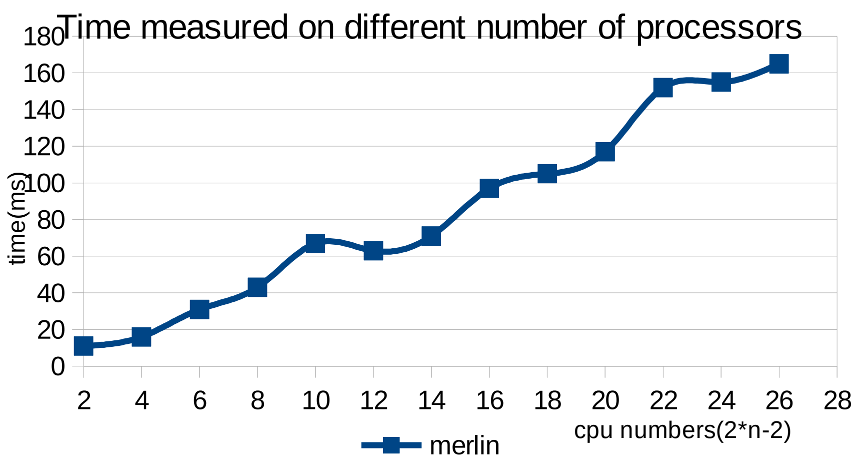

Chart 1