Project 1: DFS through parallel BFS in Directed Graphs

Project's summary

Given a graph, Depth First Searches (DFSs) are intrinsically sequential algorithms, thus to parallelize them, they are often substituted by Breadth First Searches (BFSs). Unfortunately, to gather the same information more than one BFS is often required. This project asks for a parallel implementation of a DFS as a sequence of 4 BFSs.

Problem Definition

Let G = (V, E) be a directed graph where V represents the set of vertices and E the set of directed edges.

Standard Depth-First Search (DFS) approaches traverse graphs moving "deeper" in the graph whenever possible. DFS explores edges out of the most recently discovered vertex $v \in V$ that still has unexplored edges leaving it. Once all of the v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered. This process continues until all vertices that are reachable from the original source vertex have been discovered. Besides coloring vertices and creating a depth-first forest, depth-first search also timestamps each vertex. Usually each vertex $v \in V$ receives two timestamps:

- The first timestamp (the discovery time) records when v is first discovered (and grayed)
- The second timestamp (the finishing or releasing time) records when the search finishes examining v's adjacency list.

These timestamps give important information about the structure of the graph and its characteristics. For that reason, computing them is mandatory in many applications.

Unfortunately, standard DFS-based labeling approaches traverse graphs recursively following a specific vertex order for each node. Moreover, they update global variables (which need to be protected and accessed in mutual exclusion in parallel environments) representing timestamps. These characteristics are conspicuous limitations for parallelization.

However, while DFS explores path in-depth until all vertices have been reached, BFS expands the current frontier (i.e., the current set of vertices) in parallel. In other words, BFS has no restrictions against the number of vertices being processed concurrently, and it allows a higher degree of data parallelism.

The core of the project is to substitute a DFS visit with 4 parallel BFSs visits to generate a proper graph labeling, The core idea is to realize that in a Directed Tree finding the post-order of a node is equivalent to computing an offset based on the number of nodes to the left and below the node itself. The overall process is represented by the next figure.

Fig. (a) shows an initial graph G, i.e., a very simple DAG, composed by only 7 nodes for the sake of simplicity. Fig. (b) shows the DT derived from the graph of Fig. (a). Generally speaking, there are many methods to select a parent for each vertex and transform a DAG into a DT. The so called "path based method" follows an intuitive approach, as it iterates over the graph's nodes through a top down BFS and it assigns to each children its parent's path, unless the children has already been provided with a path. In that case a node-by-node comparison of the two paths takes

place, until a decision point is found and solved by selecting the path with the “smaller” node according to the ordering relationship in the graph. Our first BFS visit perform this step.
In Fig. (c) we represent the sub-graph size for every node of the DT of Fig. (b). The algorithm to compute these values is trivial and it is again a further BFS visit.
Fig. (d) and (e) show the computation of the post-order times for all nodes. The algorithm proceeds in breadth-first, visiting the DT top down. Let us consider a vertex p and refer to its children with C_p. Among these children there is an ordering relationship given by the original DFS visit. For each vertex $v \in C_p$, we define

$$\xi_v = \sum_{i<v, i \in Cp} \zeta_i$$

which indicates the number of nodes that can be reached from all the children of the parent p of vertex v, coming before v in the ordering relationship given by a DFS. After that, as in DT there is a unique path that goes from the root r to a node v, we define

$$\tau_v = \sum_{u \in \{r \to v\}} \xi_u$$

Given ξ_v and τ_v (as computed by the previous two equations) for each vertex v, we can compute the post-order time as follows:

$$post - order = (\zeta_i - 1) + \tau_v$$

Fig. (d) indicates for each node v the values of ξ_v and τ_v as previously computed. Fig. (e) reports for all vertex v the computations of the last equation to evaluate the final post-order raking times. Finally, we compute all node’s actual label pair, i.e., $L_v = [s_v; e_v]$. It essentially proceeds breadth-first on the DT, visiting it bottom-up. Each outer-rank e_v is increased by 1 to obtain its final value. Each inner rank s_v is computed in the following way. If the vertex is a leaf, then $s_v = e_v$. If the vertex is not a leaf, s_v is equal to the minimum value of the post-order rank ev among the descendants of v. The final labels are represented in Fig. (f).

Further details on the overall method are given in the referenced paper.

Objectives

The target of this work is to write a concurrent application able to:

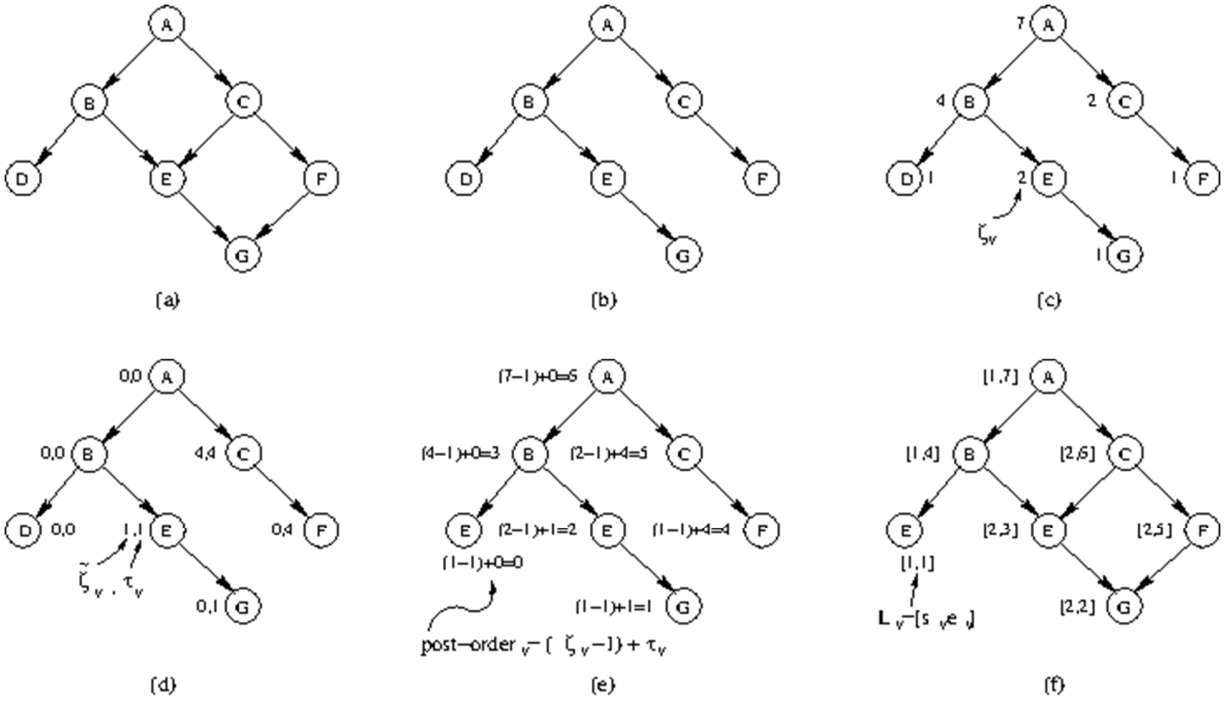
- Receive two parameters on the command line
  `file1  file2`
- Read a DAG from file `file1` (with a specific and given format).
- Generate a labeling for each graph vertex (generation phase).
- Store this labeling in the file `file2` (with the format: `vertex-id` s_v e_v).

The generation has to be performed using 4 BFSs implemented with concurrency. The result must be checked for correctness agaist an standard sequential DFS.

Required Background

Background aspects are all covered within the following classes

- Programming classes (e.g., “Algorithm and Programming”), i.e., graph theory and visits (BFS and DFS)
- “System and Device Programming” course, i.e., concurrent programming.



Working Environment

The work group can be taken with three different flavors, implementing the program in the:

- UNIX-like POSIX system
- Visual Studio Windows API
- C++.

To test the newly written program a random generator for graphs will be made available. Standard benchmarks (e.g., citeseer, cit-papents, etc.) will also be available.

Constraints

All delivered program will be checked for correctness and speed, running a small contest with (at least) a sequential-based implementation. The project is suited for groups of 3 candidates. No further constraints apply.

Contacts

Prof. Stefano Quer.

References

M. Naumov, A. Vrielink, and M. Garland, “Parallel Depth-First Search for Directed Acyclic Graphs,” in Nvidia Technical Report NVR-2017-001, 2017