Design and implementation of parallel breadth-first search

Project Report

Academic year 2020-2021
Submitted for the examination of Parallel and distributed systems: paradigms and models

By

Giuseppe Grieco
g.grieco6@studenti.unipi.it

Department of Computer Science
UNIVERSITY OF PISA, ITALY

July, 2021

Contents

1 Introduction

1.1 Problem statement

The bread-first search is an algorithm visiting the graph in amplitude. It starts from a node, often called the root or source node, and continues visiting all its descendants level by level, whereas the i-th level will contain all the nodes at a distance i from the root. For each node visited the algorithm checks its label to count the occurrences of the target label. The assumption underlying all the work is that the input to the problem is a direct and acyclic graph. In addition, for the sake of clarity, the notation used is summarized below:

Let $\mathcal{G} = (V, E)$, $|V| = n$ is the number of node and $|E| = m$ is the number of edges. For all the node $v \in V$:

- $\mathcal{N}(v) = \{u : (u, v) \in E\}$ is the neighborhood of v;
- $k_{in}(v) = |\{e : e = (u, v) \in E\}|$ is the in-degree of v;
- $k_{out}(v) = |\mathcal{N}(v)|$ is the out-degree of v;
- $k(v) = k_{in}(v) + k_{out}(v)$ is the degree of v.
- $d(u, v)$ is the distance from u to v, i.e. the minimum path from u to v.

Using the node-focus notation above, it is possible to define general properties for the graph:

- $\bar{k}$ is the average degree;
- $\bar{d}$ is the average distance;
- d_{max} is the diameter.

1.2 Preliminary analysis

1.2.1 Data structure

Before entering in the algorithmic details let's first introduce the data structure used. There are many ways to represent a graph among these the main ones are adjacency list and adjacency matrix. The choices among them is mainly a matter of the usage of the adjacency information and the expected nature of the graph. As for every node v of the graph, induced by the root, it will be necessary to go through each node $u \in \mathcal{N}(v)$, the adjacency list is way more efficient since for each node v the listing of $\mathcal{N}(v)$ takes a time proportional to $k_{out}(v)$, which is thus optimal. Moreover, if the expected input of the algorithm are "real" graphs than since they are very sparse, the representation as a adjacency list is way more efficient in terms of space complexity. In particular the presented version represent a graph G as a vector of nodes. The node is a pair, the first component is the label of the node, while the second component is the adjacency list containing the indices of the nodes in the graph.

1.2.2 Sequential version

The sequential version is a straightforward implementation of the problem description. It makes usage of two vector F and $\hat{F}$: the former is the frontier while the latter is used to add new nodes and therefore represents the frontier to be used in the next iteration. At the beginning, the algorithm initializes the frontier with the neighborhood of the source node and marks each child as visited. Then it marks the source node as visited and updates occurrences if needed (i.e. if the root has the searched label). After this first phase, the algorithm starts the first loop which iterates on each level checking that F is not empty. Subsequently, for each level, it goes through each node $v \in F$, updates occurrences if needed and then goes through each node $u \in \mathcal{N}(v)$. For each node u in the neighborhood, if the node u is not marked as visited, it adds u to $\hat{F}$. After having exhausted all the nodes of $F(i)$, the algorithm swaps F with $\hat{F}$ and then clears $\hat{F}$.

2 Proposed solution

2.1 Problem Review

2.2 Solution Description

2.3 Analysis and measurements

2.3.1 Build and execution

2.3.2 Pthread implementation

2.3.3 Fastflow implementation

3 Conclusion