Parallel Computing 44 (2015) 53–68

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

# Evaluation of connected-component labeling algorithms for distributed-memory systems

J. Iverson [a,b,*], C. Kamath [b], G. Karypis [a]

[a] University of Minnesota, Minneapolis, MN 55455, USA
[b] Lawrence Livermore National Laboratory, Livermore, CA 94550, USA

A R T I C L E   I N F O

*Article history:*
Received 29 September 2014
Received in revised form 20 January 2015
Accepted 16 February 2015
Available online 2 March 2015

*Keywords:*
Distributed-memory
Connected component
Scalability

A B S T R A C T

Connected component labeling is a key step in a wide-range of applications, such as community detection in social networks and coherent structure identification in massively-parallel scientific simulations. There have been several distributed-memory connected component algorithms described in literature; however, little has been done regarding their scalability analysis. Theoretical and experimental results are presented for five algorithms: three that are direct implementations of previous approaches, one that is an implementation of a previous approach that is optimized to reduce communication, and one that is a novel approach based on graph contraction. Under weak scaling and for certain classes of graphs, the graph contraction algorithm scales consistently better than the four other algorithms. Furthermore, it uses significantly less memory than two of the alternative methods and is of the same order in terms of memory as the other two.

## 1. Introduction

Due to the increasing rate at which scientists are able to gather data, graph analysis applications often require the processing power of large, distributed-memory machines. Whether this is due to the size of the problem or the computational resources required, scientists need efficient algorithms to utilize these large machines. One fundamental algorithm used in many graph analysis applications is connected component labeling, which can be used to identify and track regions of interest throughout the life of a simulation [1–3] or locating communities in social network graphs [4]. Identifying connected components is a well understood problem on serial machines and usually employs a breadth/depth first traversal technique or utilizes the union-find data structure.

Mapping these techniques to work efficiently on distributed-memory machines is not necessarily straight-forward. As a result, there is a considerable body of research associated with developing parallel connected component labeling algorithms. This research includes development of novel algorithms that are easier to parallelize [5–11], approaches to efficiently implement connected component labeling algorithms on distributed-memory systems [3,12–16], and studies of their scalability [3,15]. However, despite this prior work, various aspects related to the computational and memory complexity of these algorithms and their scalability have not been thoroughly analyzed in a unified way. Understanding how these methods perform and scale is important due to the wide range of hardware configurations in existing and emerging distributed-memory parallel architectures.

* Corresponding author at: University of Minnesota, Minneapolis, MN 55455, USA.
*E-mail addresses:* jiverson@cs.umn.edu (J. Iverson), kamath@llnl.gov (C. Kamath), karpyis@cs.umn.edu (G. Karypis).

http://dx.doi.org/10.1016/j.parco.2015.02.005
0167-8191/© 2015 Elsevier B.V. All rights reserved.

In this paper, five algorithms for computing connected component labeling on distributed-memory parallel systems are investigated. Of these five methods, three are direct implementations of previously published algorithms, one is an implementation based on a previous method that is optimized to reduce communication overhead, and the last is a novel method, based on repeatedly contracting the input graph at successive steps of a reduction process. Each of the algorithms is analyzed from an asymptotic perspective, providing parallel runtimes and memory requirements, assuming an equally distributed graph. Each of the algorithms is experimentally evaluated with a set of strong and weak scaling experiments, and these experimental results are compared against the expected theoretical performance. Under weak scaling and for certain classes of graphs, the graph contraction algorithm introduced in this work scales consistently better than the four other algorithms. Furthermore, it uses significantly less memory than two of the alternative methods and is of the same order in terms of memory as the other two.

The rest of the paper is organized as follows. In Section 2, definitions for terms and language used throughout the paper is provided. In Section 3, two serial algorithms for computing a connected component labeling are presented and analyzed. In Section 4, the framework for computing connected component labeling on distributed-memory parallel systems is introduced. In Section 5, five distributed-memory connected component algorithms are presented and their runtimes and memory requirements analyzed. In Section 6, the experimental design is outlined and in Section 7, the results of the experiments are presented along with a discussion of the findings. Finally, in Section 8, conclusions about this work is presented along with directions for future research regarding connected component labeling on distributed-memory systems.

## 2. Definitions and notation

A graph $G = (V, E)$ consists of a set of vertices $V$ and a set of edges $E$. The maximum shortest path between any two connected vertices in $G$ will be referred to as $\delta(G)$, also known as the diameter of $G$. Note that if $d_1, d_2, \ldots, d_n$ are the diameters of the subgraphs induced by the connected components of $G$, then $\delta(G) = \max(d_1, d_2, \ldots, d_n)$.

Throughout the paper, it is assumed that the connected component labeling (CCL) is being performed in a distributed-memory system made up of $P$ processes, connected together by an interconnect network. When a message is passed between two processes, the cost of starting the communication in hardware is represented by the variable $t_s$ and the cost of communicating a single word of data by $t_w$. The cost of a unit of local computation time is denoted $t_c$.

When computing the runtime of algorithms, $T_s$ is the time elapsed between the beginning and the end of the algorithm execution on a serial computer. In the case of a parallel system, the elapsed time is measured from the time that the first process starts executing until the last process stops executing and is referred to as $T_p$. A summary of these notations is presented in Table 1.

A key element to the distributed-memory CCL algorithms is the notion of the component adjacency graph (CAG), which is used to represent the connected components identified locally by the individual processes. The component adjacency graph $G_c = (V_c, E_c)$, contains a vertex for each connected component that was identified by the various processes based on their local portion of the graph. Two vertices $u_i, u_j \in V_c$ are connected, i.e., $(u_i, u_j) \in E_c$, if there is at least one edge in $G$ that connects a vertex in $G$'s connected component corresponding to $u_i$ with a vertex in $G$'s connected component corresponding to $u_j$. Section 4 contains a more detailed discussion of the CAG and its construction.

## 3. Serial algorithms for finding connected-components

There are two algorithms for efficiently finding connected components on a serial machine. The first is based on performing a breadth-first or depth-first traversal [17]. This algorithm finds the connected components by repeatedly starting a

**Table 1**
Summary of notations.

| Notation | Definition |
|---|---|
| $t_s$ | Cost of starting a communication |
| $t_w$ | Cost of communicating a single word over network |
| $t_c$ | Cost of local computation |
| $T_s$ | Serial runtime |
| $T^c$ | Execution time to build CAG |
| $T_p^c$ | Execution time to resolve global connectivity |
| $T_p$ | Parallel runtime |
| $T_o$ | Parallel overhead |
| $G_c$ | Component adjacency graph (CAG) |
| $V_c$ | Vertices of CAG, representing connected components of $G$ |
| $E_c$ | Edges of CAG, signifying at least one edge in $G$ which connects a vertex from each of the corresponding components |
| $\delta(G)$ | Max shortest path between two connected vertices in $G$ |
| $\beta$ | Number of adjacent processes |
| $\omega$ | Relationship between size of input graph and size of CAG |