Parallel Computing 47 (2015) 19–37

Contents lists available at ScienceDirect

Parallel Computing

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

Parallel heuristics for scalable community detection

Hao Lu [a], Mahantesh Halappanavar [b], Ananth Kalyanaraman [a,*]

[a] School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, United States
[b] Fundamental and Computational Sciences Directorate, Pacific Northwest National Laboratory, Richland, WA 99352, United States

ARTICLE INFO

Article history:
Available online 14 March 2015

Keywords:
Community detection
Parallel graph heuristics
Graph coloring
Graph clustering

ABSTRACT

Community detection has become a fundamental operation in numerous graph-theoretic applications. It is used to reveal natural divisions that exist within real world networks without imposing prior size or cardinality constraints on the set of communities. Despite its potential for application, there is only limited support for community detection on large-scale parallel computers, largely owing to the irregular and inherently sequential nature of the underlying heuristics. In this paper, we present parallelization heuristics for fast community detection using the *Louvain* method as the serial template. The Louvain method is a multi-phase, iterative heuristic for modularity optimization. Originally developed by Blondel et al. (2008), the method has become increasingly popular owing to its ability to detect high modularity community partitions in a fast and memory-efficient manner. However, the method is also inherently sequential, thereby limiting its scalability. Here, we observe certain key properties of this method that present challenges for its parallelization, and consequently propose heuristics that are designed to break the sequential barrier. For evaluation purposes, we implemented our heuristics using OpenMP multithreading, and tested them over real world graphs derived from multiple application domains (e.g., internet, citation, biological). Compared to the serial Louvain implementation, our parallel implementation is able to produce community outputs with a higher modularity for most of the inputs tested, in comparable number or fewer iterations, while providing absolute speedups of up to 16× using 32 threads.

1. Introduction

Community detection, or graph clustering, is becoming pervasive in the data analytics of various fields including (but not limited to) scientific computing, life sciences, social network analysis, and internet applications [1]. As data grows at explosive rates, the need for scalable tools to support fast implementations of complex network analytical functions such as community detection is critical. Given a graph, the problem of community detection is to compute a partitioning of vertices into communities that are closely related within and weakly across communities. Modularity is a metric that can be used to measure the quality of communities detected [2]. Modularity maximization is an NP-Complete problem [3] and therefore fast approximation heuristics are used in practice. One such heuristic is the Louvain method [4].

Our basis for selecting the Louvain heuristic for parallelization hinges on its increasing popularity within the user community and owing to its strengths in algorithmic and qualitative robustness. With well over 1700 citations to the original paper (as of this writing), the user base for this method has been rapidly expanding in the last few years. As network sizes

* Corresponding author.
E-mail addresses: luhowardmark@wsu.edu (H. Lu), hala@pnnl.gov (M. Halappanavar), ananth@eecs.wsu.edu (A. Kalyanaraman).

http://dx.doi.org/10.1016/j.parco.2015.03.003
0167-8191/

continue to grow rapidly into scales of tens or even hundreds of billions of edges [5], the memory and runtime limits of the serial implementation are likely to be tested. However, parallelization of this inherently serial algorithm can be challenging (as discussed in Section 4).

The parallel solutions presented in this paper (Section 5) provide a way to overcome key scalability challenges. In devising our algorithm, we factored in the need to parallelize without compromising the quality of the original serial heuristic and yet be capable of achieving substantial speedup. Where possible, we also factored in the need for guaranteeing stability in output across different platforms and programming models. The resulting algorithm, presented in Section 5.4, is a combination of heuristics that can be implemented on both shared and distributed memory machines. As demonstrated in our experimental section (Section 6), our multi-threaded implementations output results that have either a higher or comparable modularity to that of the serial method, and is able to reduce the time to solution by factors of up to 16×. These observations are supported over a number of real-world networks.

Contributions: The main contributions of this paper are:

(i) Introduction of novel and effective heuristics for parallelization of the Louvain algorithm on multithreaded architectures;
(ii) Experimental studies using 11 real-world networks obtained from varied sources including the DIMACS10 challenge website, University of Florida sparse matrix collection and biological databases; and
(iii) A thorough comparative study of the performance and related trade-offs among the different parallel heuristics along with the serial Louvain method.

2. Problem statement and notation

Let $G(V, E, \omega)$ be an undirected weighted graph, where V is the set of vertices, E is the set of edges and $\omega(.)$ is a weighting function that maps every edge in E to a non-zero, positive weight.[1] In the input graph, edges that connect a vertex to itself are allowed — i.e., (i, i) can be a valid edge. However, multi-edges are not allowed. Let the adjacency list of i be denoted by $\Gamma(i) = \{j | (i, j) \in E\}$. Let k_i denote the weighted degree of vertex i — i.e., $k_i = \sum_{j \in \Gamma(i)} \omega(i, j)$. We will use n to denote the number of vertices in G; M to denote the *number* of edges in the graph; and m to denote the sum of all edge weights — i.e., $m = \frac{1}{2}\sum_{i \in V} k_i$.

A *community* within graph G represents a (possibly empty[2]) subset of V. In practice, for community detection, we are interested in partitioning the vertex set V into an arbitrary number of *disjoint* non-empty communities, each with an arbitrary size (> 0 and $\leqslant n$). We call a community with just one element as a *singlet* community. We will use $C(i)$ to denote the community that contains vertex i in a given partitioning of V. We use the term *intra-community edge* to refer to an edge that connects two vertices of the same community. All other edges are referred to as *inter-community edges*. Let $E_{i \to C}$ refer to the set of all edges connecting vertex i to vertices in community C. And let $e_{i \to C}$ denote the sum of the edge weights for the edges in $E_{i \to C}$.

$$e_{i \to C} = \sum_{(i,j) \in E_{i \to C}} \omega(i, j) \tag{1}$$

Let a_C denote the sum of the degrees of all the vertices in community C (also referred to as *community degree*).

$$a_c = \sum_{i \in C} k_i \tag{2}$$

Modularity: Let $P = \{C_1, C_2, \ldots C_k\}$ denote the set of all communities in a given partitioning of the vertex set V in $G(V, E, \omega)$, where $1 \leqslant k \leqslant n$. Consequently, the *modularity* (denoted by Q) of the partitioning P is given by the following expression [2]:

$$Q = \frac{1}{2m}\sum_{i \in V} e_{i \to C(i)} - \sum_{C \in P}\left(\frac{a_C}{2m} \cdot \frac{a_C}{2m}\right) \tag{3}$$

Modularity is not an ideal metric for community detection and issues such as resolution limit have been identified [1,6]; a few variants of modularity definitions have also been devised [6–8]. However, the definition provided in Eq. (3) continues to be the more widely adopted version in practice, including in the Louvain method [4], and therefore, we will use that definition for this paper.

Community detection: Given $G(V, E, \omega)$, the problem of community detection is to compute a partitioning P of communities that maximizes modularity.

This problem has been shown to be NP-Complete [3]. Note that this problem is different from graph partitioning problem and its variants [9], where the number of clusters and the rough size distribution of those target clusters are known *a priori*. In the case of community detection, both quantities are unknown prior to computation. In fact they encapsulate the input properties that one seeks to discover out of the community detection exercise.

[1] If the graph is unweighted, then we treat every edge to be of weight 1.

[2] The notion of empty communities does not have practical relevance. We have intentionally defined it this way so as to make our later algorithmic descriptions easier. It is guaranteed, however, that all output communities at the end of our algorithm will be non-empty subsets.