ARTICLE IN PRESS

J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Parallel processing of filtered queries in attributed semantic graphs☆

Adam Lugowski [a,*], Shoaib Kamil [b,*], Aydın Buluç [c,*], Samuel Williams [c], Erika Duriakova [d], Leonid Oliker [c], Armando Fox [e], John R. Gilbert [a]

[a] Department of Computer Science, University of California, Santa Barbara, USA
[b] CSAIL, Massachusetts Institute of Technology, Cambridge, USA
[c] CRD, Lawrence Berkeley National Laboratory, Berkeley, USA
[d] School of Computer Science and Informatics, University College Dublin, Ireland
[e] EECS Department, University of California, Berkeley, USA

HIGHLIGHTS

- Domain-specific language for flexible filtering and customization of graph algorithms.
- Roofline performance model for high-performance graph exploration.
- Experimental demonstration of excellent performance and scaling.
- Demonstration of the generality by specializing two different graph algorithms.

ARTICLE INFO

Article history:
Received 19 November 2013
Received in revised form
6 June 2014
Accepted 17 August 2014
Available online xxxx

Keywords:
Graph analysis systems
Attributed semantic graphs
Graph filtering
Parallel computing
Knowledge discovery
Domain-specific languages
SEJITS
High-performance graph analysis

ABSTRACT

Execution of complex analytic queries on massive semantic graphs is a challenging problem in big-data analytics that requires high-performance parallel computing. In a semantic graph, vertices and edges carry *attributes* of various types and the analytic queries typically depend on the values of these attributes. Thus, the computation must view the graph through a *filter* that passes only those individual vertices and edges of interest. Previous investigations have developed Knowledge Discovery Toolbox (KDT), a sophisticated Python library for parallel graph computations. In KDT, the user can write custom graph algorithms by specifying operations between edges and vertices (*semiring operations*). The user can also customize existing graph algorithms by writing filters. Although the high-level language for this customization enables domain scientists to productively express their graph analytics requirements, the customized queries perform poorly due to the overhead of having to call into the Python virtual machine for each vertex and edge.

In this work, we use the Selective Embedded Just-In-Time Specialization (SEJITS) approach to automatically translate semiring operations and filters defined by programmers into a lower-level efficiency language, bypassing the upcall into Python. We evaluate our approach by comparing it with the high-performance Combinatorial BLAS engine and show that our approach combines the benefits of programming in a high-level language with executing in a low-level parallel environment. We increase the system's flexibility by developing techniques that provide users with the ability to define new vertex and edge types from Python. We also present a new Roofline model for graph traversals and show that we achieve performance that is significantly closer to the bounds suggested by the Roofline. Finally, to further understand the complex interaction with the underlying architecture, we present an analysis using performance counters that quantifies the improvement in hardware behavior in the context our SEJITS methodology. Overall, we demonstrate the first known solution to the problem of obtaining high performance from a productivity language when applying graph algorithms selectively on semantic graphs with hundreds of millions of edges and scaling to thousands of processors for graphs.

☆ This paper is the extended version of the conference paper "High-productivity and high-performance analysis of filtered semantic graphs" that was presented at the 2013 IEEE International Parallel & Distributed Processing Symposium.

* Corresponding authors.
E-mail addresses: alugowski@cs.ucsb.edu (A. Lugowski), skamil@mit.edu (S. Kamil), abuluc@lbl.gov (A. Buluç).

http://dx.doi.org/10.1016/j.jpdc.2014.08.010
0743-7315/

1. Introduction

Large-scale graph analytics is a central requirement of bioinformatics, finance, social network analysis, national security, and many other fields that deal with "big data". Going beyond simple searches, analysts use high-performance computing systems to execute complex graph algorithms on large corpora of data. Often, a large semantic graph is built up over time, with the graph vertices representing entities of interest and the edges representing relationships of various kinds—for example, social network connections, financial transactions, or interpersonal contacts.

In a semantic graph, edges and/or vertices are labeled with *attributes* that might represent a timestamp, a type of relationship, or a mode of communication. An analyst (i.e. a user of graph analytics) may want to run a complex workflow over a large graph, but wish to only use those graph edges whose attributes pass a filter defined by the analyst.

The Knowledge Discovery Toolbox [30] is a flexible, Python-based, open-source toolbox for implementing complex graph algorithms and executing them on high-performance parallel computers. KDT achieves high performance by invoking linear-algebraic computational primitives supplied by a parallel C++/MPI backend—the Combinatorial BLAS [10]. Combinatorial BLAS uses broad definitions of matrix and vector operations. The user can define custom callbacks to override the semiring scalar multiplications and additions that correspond to operations between edges and vertices.

Filters act to enable or disable KDT's action (the semiring operations) based on the attributes that label individual edges or vertices. The programmer's ability to specify custom filters and semirings directly in a high-level language like Python is crucial to ensure high-productivity and customizability of graph analysis software. This paper presents new work that allows KDT users to define filters and semirings in Python without paying the performance penalty of upcalls to Python.

Filters raise performance issues for large-scale graph analysis. In many applications it is prohibitively expensive to run a filter across an entire graph data corpus, and produce ("materialize") a new filtered graph as a temporary object for analysis. In addition to the obvious storage problems with materialization, the time spent during materialization is typically not amortized by many graph queries because the user modifies the query (or just the filter) during interactive data analysis. The alternative is to filter edges and vertices "on the fly" during execution of the complex graph algorithm. A graph algorithms expert can implement an efficient on-the-fly filter as a set of primitive Combinatorial BLAS operations coded in C/C++ and incur a significant productivity hit. Conversely, filters written at the KDT level, as predicate callbacks in Python, are productive, but incur a significant performance penalty.

Our solution to this challenge is to apply Selective Just-In-Time Specialization techniques from the SEJITS approach [12]. We define two semantic-graph-specific domain-specific languages (DSL): one for filters and one for the user-defined scalar semiring operations for flexibly implementing custom graph algorithms. Both DSLs are subsets of Python, and they use SEJITS to implement the specialization necessary for filters and semirings written in that subset to execute efficiently as low-level C++ code. Unlike writing a compiler for the full Python language, implementing our DSLs requires much less effort due to their domain-specific nature. On the other hand, our use of existing SEJITS infrastructure preserves the high-level nature of expressing computations in Python without forcing users to write C++ code.

We demonstrate that SEJITS technology significantly accelerates Python graph analytics codes written in KDT, running on clusters and multicore CPUs. An overview of our approach is shown in Fig. 1. SEJITS specialization allows our graph analytics system to bridge the gap between the performance-oriented Combinatorial BLAS and usability-oriented KDT.

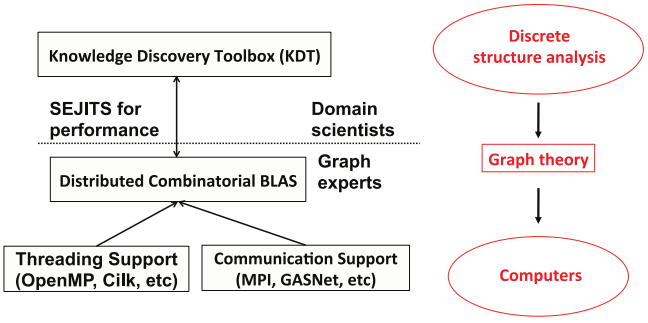

Fig. 1. Overview of the high-performance graph-analysis software architecture described in this paper. KDT has graph abstractions and uses a very high-level language. Combinatorial BLAS has sparse linear-algebra abstractions, and is geared towards performance.

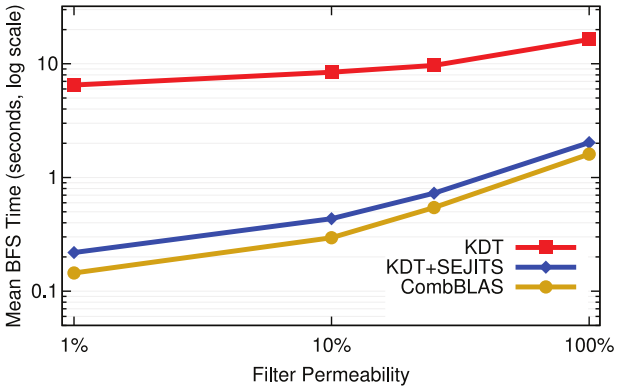

Fig. 2. Performance of a filtered BFS query, comparing three methods of implementing custom semiring operations and on-the-fly filters. The vertical axis is running time in seconds on a log scale; lower is better. From top to bottom, the methods are: high-level Python filters and semiring operations in KDT; high-level Python filters and semiring operations specialized at runtime by KDT+SEJITS (this paper's main contribution); low-level C++ filters implemented as customized semiring operations and compiled into Combinatorial BLAS. The runs use 36 cores (4 sockets) of Intel Xeon E7-8870 processors. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The primary new contributions of this paper are:

1. A domain-specific language implementation that enables flexible filtering and customization of graph algorithms without sacrificing performance, using SEJITS selective compilation techniques.
2. A new Roofline performance model [41] for high-performance graph exploration, suitable for evaluating the performance of filtered semantic graph operations.
3. Experimental demonstration of excellent performance scaling to graphs with tens of millions of vertices and hundreds of millions of edges.
4. Demonstration of the generality of our approach by specializing two different graph algorithms: breadth-first search (BFS) and maximal independent set (MIS). In particular, the MIS algorithm requires multiple programmer-defined semiring operations beyond the defaults that are provided by KDT.

Fig. 2 summarizes the work implemented in this paper, by comparing the performance of three on-the-fly filtering implementations on a breadth-first search query in a graph with 4 million vertices and 64 million edges. The chart shows time to perform the query as we synthetically increase the portion of the graph that passes the filter on an input R-MAT [28] graph of scale 22. The top,

Please cite this article in press as: A. Lugowski, et al., Parallel processing of filtered queries in attributed semantic graphs, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.08.010