The Graph BLAS effort and its implications for Exascale

David Bader, Aydın Buluç[1], John Gilbert, Joseph Gonzalez, Jeremy Kepner, Tim Mattson

Abstract

The graph abstraction provides a natural way to represent relationships among complex fast-growing scientific data sets. Graph algorithms are hard to parallelize, making their performance suboptimal on high-performance architectures. Many graph computations, however, contain sufficient parallelism that can be uncovered by using the right primitives. This position paper lays out the opportunities of defining a standard for linear-algebraic primitives for graph algorithms. Specifically, we emphasize the implications of having a Graph BLAS standard for the accessibility and availability of high-performance graph algorithms in exascale architectures.

Introduction

Data are a fundamental source of insight for experimental and computational sciences. Simulations running on exascale architectures will undoubtedly generate orders of magnitude larger data than they do today. The graph abstraction provides a natural way to represent relationships among complex fast-growing scientific data sets [2].

The Basic Linear Algebra Subprograms (BLAS) had a transformative effect on software for numerical linear algebra. With the BLAS, researchers spend less time mapping algorithms onto specific features of hardware platforms and more time on developing new algorithms. Can we define a core analogous set of mathematical primitives from which we can build most (if not all) important graph algorithms?

This is an effort to define standard building blocks for graph algorithms in the language of linear algebra. We believe that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitives.

The key insight behind this work is that when a sparse adjacency matrix represents a graph, sparse matrix-vector multiplication is the dual of breadth-first search. By generalizing the pair of operations involved in the linear algebra computations to define a semiring, we can extend the range of these primitives to support a wide range of parallel graph algorithms. A semiring consists of a set of "scalars", and two operations called "addition" and "multiplication". Speaking generally about graph algorithms, edge and vertex labels (or data or attributes) are the "scalars"; a vertex data aggregation operation is the "addition"; and an edge data processing operation is the "multiplication". Below is a set of example operations and their corresponding graph equivalents:

[1] Contact author: abuluc@lbl.gov, Lawrence Berkeley National Laboratory

- ***Matrix times matrix over semiring:*** breadth first search suite (single-source, multi-source, multi- partite, edge/vertex weight aggregation, and degree filtering)
- ***Sparse matrix indexing & assignment:*** sub-graph selection with edge and/or vertex filters
- ***Element-wise operations:*** edge/vertex elementary operations
- ***Apply/update:*** sub-graph union and intersection
- ***Matrix/vector reductions:*** user specified degree reductions

Commonly used semirings for graph algorithms are the ***Boolean semiring*** for graph traversals, ***min-plus (tropical) semiring*** for shortest paths, ***min-times semiring*** for finding maximal independent sets, the ***max-plus semiring*** for graph matching and network alignment [3], and the ***plus-times semiring*** for selecting subgraphs or contracting vertices to form quotient graphs.

Similar abstractions employed by graph algorithm packages include the generalized matrix-vector multiplication of Pegasus [7], the GAS (gather-apply-scatter) primitive of PowerGraph [8] and the "think like a vertex" approach of Pregel [9].

The Case for Linear Algebraic Primitives

Implementations of graph algorithms on parallel computers are not as scalable as their counterparts in numerical linear algebra due to several reasons. The low arithmetic intensity of common graph algorithms and their fine-grained access patterns are well acknowledged [5]. Graph algorithms also suffer from unpredictable communication patterns because they are mostly data driven. By contrast, linear algebra operations have fixed communication schedules that are built into the algorithm. Although sparse matrices are not a panacea for irregular data dependencies, the operations on them can be restructured to provide more opportunities for optimizing the communication schedule such as overlapping communication with computation and pipelining. The computation time is dominated by the latency of fetching the remote data to local registers, due to fine-grained data accesses of graph computations. Sparse matrix operations have coarse-grained parallelism, which is much less affected by latency costs.

A wide range of graph algorithms can be implemented using generalized semiring operations on sparse matrices [1]. An implementation of this approach is found in the Combinatorial BLAS [4], demonstrating that operations can be effectively implemented in a parallel. Functions offered in Combinatorial BLAS provide a reasonable basis for linear-algebraic primitives for graphs.

Implications for Exascale

There are several approaches under consideration for building an exascale computer in the 2020 time frame. The details between these different projects vary considerably, but there are some common themes. First, driven by a need to keep the power under control, the system will consist of a hundreds of thousands or even millions of processing elements. These will be organized in a hierarchical scheme exposing considerable complexity for software developers to balance memory traffic, communication and computing.

This complexity will make software for exascale systems extremely difficult to craft. On a more fundamental level, however, the large numbers of components in the system results in a mean

time to failure of some portion of the exascale computer to be small compared to the runtime of a typical application. Hence, software must be able to adapt to failure in the system and continue towards a solution even as components within the system fail.

Graph algorithms will need to be redesigned to address these joint issues of complexity, resilience, and scalability. The translation of every individual graph algorithm to exascale platforms will be prohibitively difficult due to the complexity of hardware architectures and the diversity of graph operations. Primitives allow algorithm designers to think on a higher level of abstraction, and reduce duplication of implementation efforts. Our conjecture is that with the large body of experience with sparse linear algebra in extreme scale computing, the basic graph algorithm primitives we have proposed [6] will be an effective way to manage the exascale challenge. In particular, if we can manage the exascale issues (scalability and reliability) inside the primitives, then graph algorithm developers can explore algorithms for these machines using the same approaches used on modest sized parallel systems. This would help assure that the graph-based software needed for these machines exists as exascale computers emerge.

References

[1] Kepner, Jeremy, and Gilbert, John. eds. Graph algorithms in the language of linear algebra. Vol. 22. SIAM, 2011.

[2] Chen, Fang, et al. "Topology analysis of time-dependent multi-fluid data using the Reeb graph." Computer Aided Geometric Design 30.6 (2013): 557-566.

[3] Bayati, Mohsen, et al. "Message-passing algorithms for sparse network alignment." ACM Transactions on Knowledge Discovery from Data (TKDD) 7.1 (2013): 3

[4] Buluç, Aydın, and Gilbert, John. "The Combinatorial BLAS: Design, implementation, and applications." International Journal of High Performance Computing Applications 25.4 (2011).

[5] Lumsdaine, Andrew, et al. "Challenges in parallel graph processing." Parallel Processing Letters 17.01 (2007): 5-20.

[6] Tim Mattson, David Bader, Jon Berry, Aydın Buluç, Jack Dongarra, Christos Faloutsos, John Feo, John Gilbert, Joseph Gonzalez, Bruce Hendrickson, Jeremy Kepner, Charles Leiserson, Andrew Lumsdaine, David Padua, Stephen Poole, Steve Reinhardt, Mike Stonebraker, Steve Wallach, Andrew Yoo, "Standards for Graph Algorithm Primitives", IEEE High Performance Extreme Computing Conference (2013).

[7] Kang, U., Charalampos E. Tsourakakis, and Christos Faloutsos. "Pegasus: A peta-scale graph mining system implementation and observations." ICDM'09. IEEE International Conference on Data Mining. 2009.

[8] Gonzalez, Joseph E., et al. "Powergraph: Distributed graph-parallel computation on natural graphs." Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2012.

[9] Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010