Push-Pull on Graphs is Column- and Row-based SpMV
Plus Masks

Carl Yang
University of California, Davis
Lawrence Berkeley National Laboratory
tcyang@ucdavis.edu

Aydin Buluç
Lawrence Berkeley National Laboratory
abuluc@lbl.gov

John D. Owens
University of California, Davis
jowens@ece.ucdavis.edu

ABSTRACT
Push-pull, also known as direction-optimized breadth-first-search (DOBFS), is a key optimization for making breadth-first-search (BFS) run efficiently. Linear algebra-based frameworks have advantages in conciseness, performance and portability. However, there is no work in literature describing how to implement it within a linear algebra-based framework. Our work shows that DOBFS fits well within the linear algebra-based framework.

KEYWORDS
spare matrix multiplication, breadth-first search, graph algorithms

1 INTRODUCTION
Push-pull, also known as direction-optimized breadth-first-search (DOBFS), is a key optimization for making breadth-first-search (BFS) run efficiently [3]. According to the Graph Algorithm Platform [2], no fewer than 32 out of the top 37 entries on the Graph500 benchmark (a suite for ranking the fastest graph frameworks in the world) use direction-optimizing BFS. Since its discovery, it has been extended to other traversal-based algorithms [5, 11]. One of our contributions in this paper is factoring Beamer’s direction-optimized BFS into 3 separable optimizations, and analyzing them independently—both theoretically and empirically—to determine their contribution to the overall speed-up. This allows us to generalize these optimizations to other graph algorithms, as well as fit it neatly into a linear algebra-based graph framework. These 3 optimizations are, in increasing order of specificity:

1. Change of direction: Use the push direction to take advantage of knowledge that the frontier is small, which we term input sparsity. When the frontier becomes large, go back to the pull direction.
2. Masking: In the pull direction, there is an asymptotic speed-up if we know a priori the subset of vertices to be updated, which we term output sparsity.
3. Early-exit: In the pull direction, once a single parent has been found, the computation for that undiscovered node ought to exit early from the search.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICPP ’18, Eugene, Oregon
© 2018 ACM. 123-4567-24-567/08/06 ... $15.00
DOI 0000/0000

GraphBLAS is an effort by the graph analytics community to formulate graph algorithms as sparse linear algebra [6]. The goal of the GraphBLAS API specification is to outline the common, high-level operations such as vector-vector inner product, matrix-vector product, matrix-matrix product, and define the standard interface for scientists to use these functions in a hardware-agnostic manner. This way, the runtime of the GraphBLAS implementation can make the difficult decisions about optimizing each of the GraphBLAS operations on a given piece of hardware.

Previous work by Beamer et al. [4] and Besta et al. [5] have observed that push and pull correspond to column- and row-based matvec (Optimization 1). However, this realization has not made it into the sole GraphBLAS implementation in existence so far, namely SuiteSparse GraphBLAS [9]. In SuiteSparse GraphBLAS, the BFS executes in only the forward (push) direction.

The key distinction between our work and that of Shun and Blelloch [11], Besta et al. [5], and Beamer et al. [4] is that while they take advantage of input sparsity using change of direction (Optimization 1), they do not analyze using output sparsity through masking (Optimization 2), which we show theoretically and empirically is critical for high performance. Furthermore, we submit this speed-up extends to all algorithms for which there is a priori information regarding the sparsity pattern of the output such as triangle counting [1], adaptive PageRank [10], batched betweenness centrality [6], and maximal independent set [7].

Since the input vector can be either sparse or dense, we refrain from referring to this operation as SpMSpV (sparse matrix-sparse vector) or SpMV (sparse matrix-dense vector). Instead, we will refer to it as matvec (short for matrix-vector multiplication and known in GraphBLAS as GrB_mvx).

2 APPLICATIONS
The details of these optimizations are in the full paper of this conference [12], so we will give some details here on how masking can be used in the four applications mentioned above.

2.1 Triangle counting
Algorithm 1 gives the algorithm for triangle counting as described by Azad, Buluç and Gilbert [1].

2.2 Adaptive PageRank
Algorithm 2 gives the algorithm for adaptive PageRank as described by Kamwar, Haveliwala and Golub [10].
Algorithm 1 Triangle counting with A as the mask.

1: procedure TRIANGLECOUNT(Undirected graph A) \(\triangleright\) tril() in MATLAB
2:  L ← lower_triangular(A)
3:  U ← upper_triangular(A) \(\triangleright\) triu() in MATLAB
4:  C ← A \(\times\) (L \(\times\) U)
5:  num_tri ← nnz(C)
6:  return num_tri
7: end procedure

Algorithm 2 Adaptive PageRank with v as the mask.

1: procedure ADAPTIVEPR(Directed graph A, Initial PageRank \(x^{(0)}\), Absolute tolerance \(\varepsilon\))
2:  v ← \{1, 1, ..., 1\}
3:  k ← 0
4:  while \(\delta < \varepsilon\) do
5:    \(x^{(k+1)}\) ← v \(\ast\) \((A \times x^{(k)})\)
6:    v ← \(x^{(k+1)} - x^{(k)}\) \(\varepsilon\)
7:    k ← k + 1
8:  \(\delta = ||Ax^{(k)} - x^{(k)}||_{1}\)
9:  end while
10: return \(x^{(k)}\)
11: end procedure

2.3 Batched betweenness centrality
Algorithm 3 gives the algorithm for batched betweenness centrality as described by Buluç et al. [6].

2.4 Maximal independent set
Algorithm 4 gives an algorithm for maximal independent set as described by Buluç et al. [7].

3 CONCLUSION
In our full paper [12], we have demonstrated that push-pull corresponds to the concept of column- and row-based masked matvec. We have experimental evidence that masking is advantageous from both a theoretical and empirical standpoint. In this paper, we have demonstrated that masking is a very common operation in graph analytics. It is applicable to applications such as triangle counting, adaptive PageRank, batched betweenness centrality, and maximal independent set. A possible future research direction would be to provide experimental evidence for these four algorithms against existing, state-of-the-art implementations.

REFERENCES