LAGraph Algorithms

Various Artists

February 11, 2021

Abstract

Theoretical documentation for LAGraph.

1 Introduction

The goal of this document is to present a notation for GraphBLAS algorithms and showcase it using important GraphBLAS algorithms.

2 The GraphBLAS

Goal The goal of GraphBLAS is to create a layer of abstraction between the graph algorithms and the graph analytics framework, separating the concerns of the algorithm developers from those of the framework developers and hardware designers. To achieve this, it builds on the theoretical framework of matrix operations on arbitrary semirings [3], which allows defining graph algorithms in the language of linear algebra [4]. To ensure portability, the GraphBLAS standard defines a C API that can be implemented on a variety of hardware including GPUs.

Data structures A graph with n vertices can be stored as a square adjacency matrix $\mathtt{A} \in \mathtt{uint}^{n \times n}$, where rows and columns both represent vertices of the graph and element $A(i, j)$ contains the number of edges from vertex i to vertex j. If the graph is undirected, the matrix is symmetric.

Navigation The fundamental step in GraphBLAS is the multiplication of an adjacency matrix with another matrix or vector over a selected semiring. For example, the operation $\mathtt{HasMember}\,\mathtt{lor.land}\,\mathtt{IsLocatedIn}$ computed over the *"logical or.logical and"* semiring returns a matrix representing the Places where a Forum's members are located in. Meanwhile, when computed over the conventional arithmetic *"plus.times"* semiring, $\mathtt{HasMember} +.\times \mathtt{IsLocatedIn}$ also returns the number of such Persons. A traversal from a certain set of vertices can be expressed by using a boolean vector $\mathtt{f}$ (often referred to as the *frontier*, *wavefront*, or *queue*) and setting `true` values for the elements corresponding to source vertices. For example, for Forums $\mathtt{f} \in \mathtt{bool}^{|forums|}$, $\mathtt{f}\,\mathtt{lor.land}\,\mathtt{HasMember}$ returns the Persons who belong to any of the forums in `f`. The BFS navigation step can also be captured using other semirings such as `lor.first`, where $\mathtt{first}(x, y) = x$; `lor.second`, where $\mathtt{second}(x, y) = y$; and `any.pair`, where $\mathtt{any}(x, y)$ returns either x or y, and $\mathtt{pair}(x, y) = 1$ [2].

2.1 Notation

Table 1 contains the notation of GraphBLAS operations Additionally, we use $\mathtt{D} = \mathtt{diag}(\mathtt{j}, \mathtt{n})$ to construct a diagonal matrix $\mathtt{D} \mathrel{<\!\!-} \{\mathtt{j}, \mathtt{j}, [1, 1, \ldots, 1]\}$. The elements of the matrix are $\mathtt{D}(\mathtt{j}, \mathtt{j}) = 1$ for $\mathtt{j} \in \mathtt{j}$.

2.1.1 Masks

Masks $\mathtt{C} \langle \mathtt{M} \rangle$ and $\mathtt{u} \langle \mathtt{m} \rangle$ are used to selectively write to the result matrix/vector. The complements of the masks can be selected with the negation symbol, denoted with: $\mathtt{C} \langle \mathtt{!M} \rangle$ and $\mathtt{u} \langle \mathtt{!m} \rangle$, respectively.

Masks with "replace" semantics (annihilating all elements outside the mask) are denoted with

- $\mathtt{C} \langle\langle \mathtt{M} \rangle\rangle$
- $\mathtt{C} \langle\langle \mathtt{!M} \rangle\rangle$
- $\mathtt{u} \langle\langle \mathtt{m} \rangle\rangle$
- $\mathtt{u} \langle\langle \mathtt{!m} \rangle\rangle$

The structure of the mask is denoted with:

op./method	name	notation
mxm	matrix-matrix multiplication	$C <M> + = A + . \times B$
vxm	vector-matrix multiplication	$w <m> + = u + . \times A$
mxv	matrix-vector multiplication	$w <m> + = A + . \times u$
eWiseAdd	element-wise addition	$C <M> + = A + B$
	set union of patterns	$w <m> + = u + v$
eWiseMult	element-wise multiplication	$C <M> + = A \times B$
	set intersection of patterns	$w <m> + = u \times v$
extract	extract submatrix	$C <M> + = A(i, j)$
	extract column vector	$w <m> + = A(:, j)$
	extract row vector	$w <m> + = A(i, :)$
	extract subvector	$w <m> + = u(i)$
assign	assign matrix to submatrix with mask for C	$C <M> (i, j) + = A$
	assign scalar to submatrix with mask for C	$C <M> (i, j) + = s$
	assign vector to subvector with mask for w	$w <m> (i) + = u$
	assign scalar to subvector with mask for w	$w <m> (i) + = s$
subassign (GxB)	assign matrix to submatrix with submask for C(i, j)	$C(i, j) <M> + = A$
	assign scalar to submatrix with submask for C(i, j)	$C(i, j) <M> + = s$
	assign vector to subvector with submask for w(i)	$w(i) <m> + = u$
	assign scalar to subvector with submask for w(i)	$w(i) <m> + = s$
apply	apply unary operator	$C <M> + = f(()A)$
		$w <m> + = f(()u)$
select (GxB)	apply select operator	$C <M> + = select(A, f(()k))$
		$C <M> + = select(low \leq A \leq up)$
		$w <m> + = select(u, f(()k))$
		$w <m> + = select(low \leq u \leq up)$
reduce	reduce matrix to column vector	$w <m> + = [+ A]$
	reduce matrix to scalar	$s + = [+ A]$
	reduce vector to scalar	$s + = [+ u]$
transpose	transpose	$C <M> + = A'$
kronecker	Kronecker multiplication	$C <M> + = kron(A, B)$
new	new matrix	$A = \mathsf{TYPEPRECISION}(n, m)$
	new vector	$u = \mathsf{TYPEPRECISION}(n)$
build	matrix from tuples	C <- $\{i, j, x\}$
	vector from tuples	w <- $\{i, x\}$
extractTuples	extract index/value arrays	$\{i, j, x\}$<- A
		$\{i, x\}$<- u
dup	duplicate matrix	C<- A
	duplicate vector	w<- u
extractElement	extract scalar element	$s = A(i, j)$
		$s = u(i)$
setElement	set element	$C(i, j) = s$
		$w(i) = s$

Table 1: GraphBLAS operations and methods based on [1]. *Notation:* Matrices and vectors are typeset in bold, starting with uppercase (A) and lowercase (u) letters, respectively. Scalars including indices are lowercase italic (k, i, j) while arrays are lowercase bold italic (x, i, j). + and × are the addition and multiplication operators forming a semiring and default to conventional arithmetic + and × operators. + is the accumulation operator.

- $\texttt{C} < \{\texttt{M}\} >$
- $\texttt{C} < !\{\texttt{M}\} >$
- $\texttt{u} < \{\texttt{m}\} >$
- $\texttt{u} < !\{\texttt{m}\} >$

Combining structure and replace semantics is possible:

- $\texttt{C} \ll \{\texttt{M}\} \gg$
- $\texttt{C} \ll !\{\texttt{M}\} \gg$
- $\texttt{u} \ll \{\texttt{m}\} \gg$
- $\texttt{u} \ll !\{\texttt{m}\} \gg$

Initializing scalars, vectors, and matrices (GraphBLAS methods):

- $\texttt{s} = \texttt{fp64}()$
- $\texttt{u} = \texttt{fp32}(\texttt{n})$
- $\texttt{A} = \texttt{uint16}(\texttt{m}, \texttt{n})$
- $\texttt{A} = \texttt{int64}(\texttt{k}, \texttt{m})$

3 Algorithms

LAGraph [5] implements graph algorithms using the GraphBLAS C API [6].

Here are a few algorithms that could be included in this document:

```
Input: A, n, startVertex
Function BFS
    f(startVertex) = T
    for level = 1 to n − 1 do
        s <f>= level
        f <<!s>>= fA
```

```
Input: A, n, startVertex
Function ParentsBFS
    f(startVertex) = 0
    for level = 1 to n − 1 do
        s <f>= f
        f <<!s>>= f any.firstj1 A
```

```
Input: A, A', n, startVertex
Function DirectionOptimizingBFS
    f(startVertex) = T
    for level = 1 to n − 1 do
        s <f>= level
        if Push(A, f) then
            f <<!s>>= fA
        else
            f <<!s>>= A'f
```

```
Input: A, n, startVertices
Function ConcurrentBFS
    F = diag(startVertices, $n$)
    for level = 1 to $n - 1$ do
        S <F>= level
        F <<!S>>= FA
```

Figure 1: Sketch algorithms of BFS variants. The Push(()A, f) function makes a decision on whether it is cheaper to push or pull using heuristics based on the sparsity of the frontier f and adjacency matrix A.

Algorithm 1: Multi-source breadth-first search.

```
Data: ...
Result: ...
Function MSBFS
    Frontier = diag(sources, n)
    for level = 1 to $n - 1$ do
        Seen <Frontier>= level
        Frontier <<!Seen>>= Frontier any.pair A
```

Algorithm 2: All-pairs shortest distance (on undirected, unweighted graphs) [7].

```
Data: ...
Result: ...
Function $APD$(A, n, deg)
    Z = A
    Z += A +.× A
    B = {select(Z, offdiag)} // use the pattern as a Boolean matrix
    if A == B then
        return A
    T = $APD$(B, n, deg)
    X = T +.× A
    Tscaled = T +.× diag(()deg)
    Xfiltered = select(X, X < Tscaled)
    return (2 × T) MINUS Xfiltered
Function $APD$(A)
    deg = [+A]
    Distance = $APD$(A, n, deg)
    sp = [+Distance]
```

Algorithm 3: Betweenness centrality.

```
Function $MSBFS$
    // The NumSp structure holds the number of shortest paths for each node and starting
       vertex discovered so far.
    // Initialized to source vertices.
    NumSp <- {s, [1, 1, ..., 1]}
    // The Frontier holds the number of shortest paths for each node and starting vertex
       discovered so far.
    // Initialized to source vertices.
    Frontier <NumSp>= A(s, :)
    d = 0
    // The Sigmas matrices store frontier information for each level of the BFS phase.
    // BFS phase (forward sweep)
    do
        // Sigmas[d](:, s) = d^th level frontier from source vertex s
        Sigmas[d] = bool(n, nsver)
        Sigmas[d](:, :) = Frontier                        // Convert matrix to Boolean
        NumSp = NumSp + Frontier                             // Accumulate path counts
        Frontier <<NumSp>>= A' +.× Frontier                      // Update frontier
    while nvals(Frontier) > 0
    NumSpInv = fp32(n, nsver)
    NumSpInv = 1.0 DIV NumSp
    BCU = fp32(n, nsver)
    BCU(:) = 1.0                    // Make BCU dense, initialize all elements to 1.0
    W = fp32(n, nsver)
    // Tally phase (backward sweep)
    for i = d − 1 downto 0 do
        W <<Sigmas[i]>>= NumSpInv DIV BCU
        W <<Sigmas[i − 1]>>= A +.× W  // Add contributions by successors and mask with that
         BFS level's frontier.
        BCU += W × NumSp
    // Row reduce BCU and subtract nsver from every entry to account for 1 extra value
       per BCU row element
    delta = [+BCU]
    delta MINUS = nsver
```

Algorithm 4: PageRank (used in Graphalytics).

```
Data: alpha constant (damping factor)
Result: ...
Function PageRank
    pr(:) = $1/n$
    outdegrees = $[+_{j}$ A(:, j)$]$
    for k = 1 to numIterations do
        importance = pr DIV outdegrees
        importance = times(()importance, alpha)        // apply the times$(()x, s) = x \cdot s$ operator
        importance = importance $+ . \times$ A
        danglingVertexRanks < !outdegrees >= pr(:)
        totalDanglingRank = (alpha)/(n) $\times$[+ danglingVertexRanks]
        pr = (1 $-$ alpha)/(n) + totalDanglingRank
        pr = pr + importance
```

Algorithm 5: Algebraic Bellman-Ford.

```
Function SSSP
    d(s) = 0
    for k = 1 to n $-$ 1 do
        $d'$ = d min.plus A
        if $d'$ == d then break
        d <- $d'$
```

Algorithm 6: Delta-stepping SSSP.

```
Data:
  $A, A_H, A_L \in$ fp(|V|, |V|)
  s, i $\in$ uint()
  $\Delta \in$ fp()
  $t, t_{Req} \in$ fp(|V|)
  $t_{B_i}, e \in$ uint(|V|)
Function DeltaStepping
    $A_L$ = select($0 < A \leq \Delta$)
    $A_H$ = select($\Delta < A$)
    t(:) = $\infty$
    t(s) = 0
    while nvals(select($i\Delta \leq t$)) $\neq 0$ do
        s = 0
        $t_{B_i}$ = select($i\Delta \leq t < (i+1)\Delta$)
        while $t_{B_i} \neq 0$ do
            $t_{Req} = A_L$' $+ . \times$ ($t \times t_{B_i}$)
            e = select($0 < e + t_{B_i}$)
            $t_{B_i}$ = select($i\Delta \leq t_{Req} < (i+1)\Delta$) $\times$($t_{Req} \underset{+}{\min} t$)
            $t_{B_i}$ = select($i\Delta \leq t_{Req} < (i+1)\Delta$) $\times$($t_{Req} +_{\min} t$)
            $t_{B_i}$ = select($i\Delta \leq t_{Req} < (i+1)\Delta$) $\times$($t_{Req} \min_{+} t$)
            t = t min $t_{Req}$
        $t_{Req} = A_H$' $+ . \times$ ($t \times e$)
        t = t min $t_{Req}$
        i = i + 1
```

Algorithm 7: All-pairs shortest path (Floyd–Warshall algorithm).

```
Function FloydWarshall
    D <- A
    for k = 1 to n do
        D = D min[D(:, k) min.plus (D(k, :)]
```

Algorithm 8: FastSV algorithm.

```
Function FastSV
    n = nrows(A)
    gf = f
    dup = gf
    mngf = gf
    {i, x} <- f
    repeat
        // Step 1: Stochastic hooking
        mngf = mngf min A
        mngf = mngf second.min gf
        f(x) = f min mngf
        // Step 2: Aggressive hooking
        f = f min mngf
        // Step 3: Shortcutting
        f = f min gf
        // Step 4: Calculate grandparents
        {i, x} <- f
        gf = f(x)
        // Step 5: Check termination
        diff = dup $\neq$ gf
        sum = $[+_i$ diff(i)$]$
        dup = gf
    until sum == 0
```

Algorithm 9: Triangle count (Cohen's algorithm).

```
Function TriangleCount
    L = tril(A)
    U = triu(A)
    B = L +.× U
    C = B × A
    t = [+C]/2
```

Algorithm 10: Triangle count (Sandia).

```
Function TriangleCount
    L = tril(A)
    C <L>= L +.× L
    t = [+C]
```

Algorithm 11: Triangle count (FLAME).

```
Function TriangleCountFlame
    for i = 2 to n − 1 do
        $A_{20}$ = A(i + 1 : n, 0 : i − 1)
        $a_{10}$ = A(0 : i − 1, i)
        $a_{12}$ = A(i, i + 1 : n)
        t += $a_{10}$ +.× $A_{20}$ +.× $a_{12}$
```

Algorithm 12: Local clustering coefficient.

```
Function PageRank
    Tri <A>= A +.× A            // compute triangle count matrix
    tri = [+Tri]                // reduce to triangle count vector
    deg = [+A]                  // reduce to vertex degree vector
    wed = perm2(()deg)          // apply perm2(()$x$) = $x \cdot (x-1)$ to get wedge count vector
    lcc = tri DIV wed           // LCC vector
```

Algorithm 13: *k*-truss algorithm.

```
Function KTruss
    C <- A
    nonzeros <- nvals($C$)
    for $i = 1$ to $n - 1$ do
        C <C>= C +.land C
        C = select(C $\geq$ k $-$ 2)
        if nonzeros == nvals($C$) then
            break
        nonzeros <- nvals($C$)
```

Algorithm 14: Louvain algorithm (WIP).

```
Function Louvain
    G += G'
    k = [+ A]
    m = (1)/(2)[+ k]
    S <- I

    vertices_changed <- nvals(k)
    while vertices_changed > 0 do
        for j $\in$ range($|V|$) do
            v = G(j, :)
            $t_q$ = v any.pair S
            sr = S(j, :)
            S(j, :) = empty

            q <- k
            q <k> $\times$= $-$ k(j)/m
            q += v
            $q_1$ <$t_q$>= q +.$\times$ S

            t = ($q_1$ == [max $q_1$])
            while nvals($t$) $\neq$ 1 do
                p = random() $\times$ t
                t = (p == [max p])
            S(j, :) = t

            if nvals(sr $\times$ t) == 0 then
                vertices_changed = nvals(k)
            vertices_changed = vertices_changed $-$ 1
```

Algorithm 15: Community detection using label propagation (for undirected graphs).

```
Function CDLP
    L <- diag([0, 1, ..., n $-$ 1])
    for k = 1 to t do
        F = A any.second L                                  // Frequency matrix
        {i, _, x} <- F
        merge_sort_pairs(i, x)
        labels = for each row in i, select min mode value from x
```

References

[1] T. A. Davis, "Algorithm 1000: SuiteSparse:GraphBLAS: Graph algorithms in the language of sparse linear algebra," *ACM Trans. Math. Softw.*, 2019. [Online]. Available: https://doi.org/10.1145/3322125

[2] ——. (2020) User guide for SuiteSparse:GraphBLAS, version 3.3.3. https://people.engr.tamu.edu/davis/GraphBLAS.html.

[3] J. Kepner *et al.*, "Mathematical foundations of the GraphBLAS," in *HPEC*. IEEE, 2016. [Online]. Available: https://doi.org/10.1109/HPEC.2016.7761646

[4] J. Kepner and J. R. Gilbert, Eds., *Graph Algorithms in the Language of Linear Algebra.* SIAM, 2011. [Online]. Available: https://doi.org/10.1137/1.9780898719918

[5] T. Mattson *et al.*, "LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS," in *GrAPL at IPDPS*, 2019. [Online]. Available: https://doi.org/10.1109/IPDPSW.2019.00053

[6] T. G. Mattson *et al.*, "GraphBLAS C API: Ideas for future versions of the specification," in *HPEC*. IEEE, 2017. [Online]. Available: https://doi.org/10.1109/HPEC.2017.8091095

[7] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 400–403, 1995. [Online]. Available: https://doi.org/10.1006/jcss.1995.1078