GraphBLAS Tutorial at HPEC

Tuesday, 25 September 2018

Two 90 minute sessions: 9-10:30, 11-12:30

Roughly speaking, we can break the time into six thirty minute chunks:

## 09:00 - 09:30 Part 1. Introduction to GraphBLAS, and tooling for the hands-on portion

Materials that we need to provide (USB stick or from GraphBLAS.org website):

* The C API Specification
* The GraphBLAS library source tarball (SuiteSparse + User Guide?)
* Library binary??
* Cmake version 3+
* Hardcopy: A Qwik-Reference "card" (1 double sided sheet) with API info relevant to course
  + List of GraphBLAS operation primitives (refer to spec table 4.1?)
  + Dissection of a signature (output container, mask, accum, op, input containers, descriptor)
  + List of Domains “labels” (types)
  + List of Unary and Binary operations
  + List of Descriptor label, value pairs
  + GraphBLAS Objects
* <add more reference materials here>

Concepts to teach

* Build Environment: Can they edit compile and run a test program?
  + hpec\_ex0.cpp
* Graph - Matrix duality
* GraphBLAS operations list? GraphBLAS C API Signature dissection? (Make sure it is on Qwik Reference)
* GraphBLAS Objects
  + Matrix
  + Vector
  + Masks (as Matrix or Vector)
  + Domains?
  + Operators: UnaryOp, BinaryOp, Monoid, Semiring
  + Descriptor (maybe not)

## 09:30 - 10:00 Part 2: "Hello, World" Exercise: Build an adjacency matrix

Hands on exercise:

**6**

**4**

**3**

**2**

**1**

**5**

**7**

Given: the 7 node, 12 edge "logo" graph

* 1-based
  + Row indices: {1, 1, 2, 2, 3, 4, 4, 5, 6, 7, 7, 7}
  + Col indices: {2, 4, 5, 7, 6, 1, 3, 6, 3, 3, 4, 5}
* 0-based
  + Row indices: {0, 0, 1, 1, 2, 3, 3, 4, 5, 6, 6, 6}
  + Col indices: {1, 3, 4, 6, 5, 0, 2, 5, 2, 2, 3, 4}

Build: a GraphBLAS Matrix representing the graph, and validate that it is correct

Questions/Pit Falls:

* which Domain type, signed or unsigned integer or boolean?
* 1-based indexing in graph vs. 0-based indexing in Matrix

Functions/Concepts needed:

* Matrix\_new, Domain, dimensions
* Matrix\_build (index and value arrays) or Matrix\_setElement
* Matrix\_nrows/ncols/nvals
* Maybe: Matrix\_getElement for a pretty print function? 🡨 TGM: yes, we need this
* DON’T FORGET GrB\_free!!

Code: hpec\_ex1\_hello\_world.c

Wrap up information?

## 10:00 - 10:30 Part 3: "Finding Neighbors, How many Neighbors" Exercise

Concepts to teach:

* Semiring vs. Linear Algebra (Arithmetic/+.\*, Logical/Or.And, Min.+) 🡨 TGM: isn’t “real field” the right term, not “linear algebra”? We’re still defining a linear algebra, its just over a domain that is a semi-ring.
* GraphBLAS primitive mxv (“find neighbors”)
* Matrix-vector "multiply" can find neighbors

**A**

**1**

**3**

**2**

**4**

**5**

**6**

**7**

**4**

**5**

**6**

**7**

**3**

**2**

**1**

**to vertex**

**from vertex**

**T**

**=**

**V**

**AV**

**T**

Hands on exercise:

Given: Matrix from previous exercise, pick a source node

Compute: the set of 1-hop neighbors reachable from source

Questions/Pit Falls?

* Matrix transpose?
* Using Vector\_build instead of simpler Vector\_setElement

Code

* hpec\_ex2\_neighbors.c

Functions/Concepts needed:

* Vector\_new, Vector\_build, Vector\_setElement

Additional exercise(s):

* mxv (with transpose of A) vs. vxm (without the transpose of A)
* Primitive: degree (reduction)
  + compute the out-degree, (row) reduce(A, +)
    - {2, 2, 1, 2, 1, 1, 3}
  + compute the in-degree, (row) reduce(A’, +) (a.k.a. column reduce)
    - {1, 1, 3, 2, 2, 2, 1}
* Are there any other single operation calls that produce commonly used information?
* load the HPEC data set graph and repeat any of these?

## **10:30 – 11:00: BREAK**

## 11:00 - 11:30: Part 4: Breadth-first traversal, (iterative find-neighbor primitive with mask)

Concepts to teach:

* Overview of computing levels in a BFS (code is appendix B1 of C API Spec, 53 lines for the algorithm
* Assign primitive to set the level (1-based can be used as a mask too)
* Using a mask with structural complement (two concepts) to not revisit neighbors
* Semiring vs. Linear Algebra (Arithmetic/+.\*, Logical/Or.And, Min.+) 🡨 TGM: see linear algebra comment above

Levels from node 4 = {2, 3, 2, 1, 4, 3, 4}

Code is adapted from Appendix B.1 of the C API specification (with slight modifications to names, and replacing reduce with Vector\_nvals).

* hpec\_ex3\_level\_bfs.c

GraphBLAS methods and operations used:

* GrB\_Matrix\_nrows
* GrB\_Vector\_new, GrB\_Vector\_setElement
* GrB\_Monoid\_new
* GrB\_Semiring\_new
* GrB\_Descriptor\_new, GrB\_Descriptor\_set
* GrB\_assign (scalar-to-vector variant)
* GrB\_vxm or GrB\_mxv (with transpose) – both with complemented mask
* GrB\_Vector\_nvals (appendix uses reduce here)

## 11:30 - 12:00: Part 5: Finding parents

\*\*\*This section needs more explanation / intuition for why it works\*\*\*

In this section we could teach them how to use the API to find the parents of a given node from the level list that they just computed in the previous section.

We need to do this without given them a “parent-list” BFS that uses “index\_of” utility and MinSecond ‘semiring’ because there is just not enough time to go over all of the gorey details.

Note that:

A’ +.\* e\_i finds out-neighbors of vertex i.

Transpose (primitive) reverses direction, so…

A +.\* e\_i finds the in-neighbors of vertex i.

We are only interested in finding those in neighbors with level one less than vertex i.

The approach is to create a mask for the graph that only selects the directed edges used in the BFS traversal. This will result in a directed acyclic graph from which parent information can be extracted directly with GrB\_Matrix\_extractTuples:

We can create this mask from the level result in the previous exercise as follows:

D = diag(levels) // 1-based or 0-based does not matter

diag(levels) is performed by extracting indices and values from level and building a matrix from (indices, indices, values) tuples, OR by extracting one element at a time from levels and setting appropriate diagonal element in D. The former requires malloc of index and value arrays.

The Mask for the graph is computed as follows (**intuitive explanation needed**):

Mask = (A +.\* D) > (D +.\* A)

This is performed with two calls to GrB\_mxm, using the Arithmetic semiring, and one call to GrB\_eWiseMult using GrB\_GT\_UINT64

The BFS tree, Atree, is extracted from the graph, A, using masked assignment using GrB\_IDENTITY\_UINT64:

Atree<Mask> = A

Finally the parent list information is extracted from Atree and built into parent vector as follows:

GrB\_Matrix\_extractTuples(row\_indices, col\_indices, values, &nvals, Atree);

GrB\_Vector\_build(parents, col\_indices, row\_indices, nvals, GrB\_SECOND\_UINT64);

Pretty slick (?!)…no user-defined unary operators required.

From the parent\_list you can walk back from any node to the original BFS source vertex and build a “path”.

## 12:00 - 12:30: Part 6: Find path(s) between X and Y, and/or “Do something interesting”

Present the HPEC dataset(s):

* Co-authorship Graph:
  + 1747 authors (after entity resolution)
  + 10072 edges
* Topic modelling Graph
  + 883 papers

Provide tools to peruse the dataset

* Author lookup
* Papers by author
* Coauthors
* Top N most similar papers

Extra Credit:

* Provide a library of the algorithms from the appendix of the C API Spec
* Provide algorithms translated from GBTL

-------------------------------------------------------------------------------

# Proposed appendix … not covered in the tutorial but provided as backup

Describe techniques and code used to create the HPEC dataset. Even though we won’t cover this in the tutorial, it will be interesting for many of the students (and myself as well) to look at on their own.