Tumbling Down the GraphBLAS Rabbit Hole with SHMEM

Curtis Hughey, Department of Defense

Outline

GraphBLAS Overview

"An open effort to define standard building blocks for graph algorithms in the language of linear algebra"

Sparse graphs can be appropriate models for many applications

Small number of primitives, type abstraction, etc.

Custom semiring operations - abstracts out the "normal" multiplication and addition used in matrix/vector operations.

Huge opportunities for under-the-hood optimization: exploit semiring properties and non-blocking mode to parallelize GraphBLAS primitives

Increased overhead

GraphBLAS Primitives

Operation	Description
mxm	Matrix-matrix multiply
mxv	Matrix-vector multiply
vxm	Vector-matrix multiply
eWiseAdd	Matrix/vector addition
eWiseMult	Matrix/vector Hadamard product
reduce (row)	Row-wise matrix reduction
reduce (scalar)	Scalar matrix/vector reduction
apply	Matrix/vector function application
transpose	Matrix transpose
extract	Matrix/vector tuple extraction
assign	Matrix/vector assignment

BFS Algorithm

```
GrB_Info BFS(GrB_Vector *v,   // Output vector
             GrB_Matrix A,    // Input matrix
             GrB_Index s) {   // Root index
  // Initialize vector q, set to true at index s
  // q represents frontier vertices
  do {
    ++d;  // Next level in BFS traversal
    // v[q] = d
    GrB_assign(*v, q, GrB_NULL, d, GrB_ALL, n, GrB_NULL);

    // q[!v] = A ||.&& q
    GrB_mxv(q, *v, GrB_NULL, Boolean, A, q, desc);

    // succ = ||(q)
    GrB_reduce(&succ, GrB_NULL, Lor, q, GrB_NULL);
  } while (succ);
  return GrB_SUCCESS;
}
```

BFS Walkthrough

$$\overset{A}{\begin{bmatrix} & & & 1 & & & \\ 1 & & 1 & & & & \\ & & & 1 & & 1 & 1 \\ 1 & & & & & & 1 \\ & 1 & & & & & 1 \\ & & 1 & & 1 & & \\ & 1 & & & & & \end{bmatrix}} \cdot \overset{q}{\begin{bmatrix} \\ \\ \\ 1 \\ \\ \\ \\ \end{bmatrix}}$$

A: Adjacency matrix

q: Vertices just visited (to begin with, just the root node)

BFS Walkthrough

$$\overset{A}{\begin{bmatrix} & & & 1 & & & \\ 1 & & 1 & & & & \\ & & & 1 & & 1 & 1 \\ 1 & & & & & & 1 \\ & 1 & & & & & 1 \\ & & 1 & & 1 & & \\ & 1 & & & & & \end{bmatrix}} \cdot \overset{q}{\begin{bmatrix} \\ \\ \\ 1 \\ \\ \\ \\ \end{bmatrix}} \rightarrow \overset{q}{\begin{bmatrix} 1 \\ \\ 1 \\ \\ \\ \\ \\ \end{bmatrix}} , \overset{v}{\begin{bmatrix} 2 \\ \\ 2 \\ 1 \\ \\ \\ \\ \end{bmatrix}}$$

A: Adjacency matrix
q: Vertices just visited (to begin with, just the root node)
v: The output vector, measures distance from root node, starting from 1

Two new nodes are discovered (green)

BFS Walkthrough

$$\overset{A}{\begin{bmatrix} & & & 1 & & & \\ 1 & & 1 & & & & \\ & & & 1 & & 1 & 1 \\ 1 & & & & & & 1 \\ & 1 & & & & & 1 \\ & & 1 & & 1 & & \\ & 1 & & & & & \end{bmatrix}} \cdot \overset{q}{\begin{bmatrix} 1 \\ \\ 1 \\ \\ \\ \\ \\ \end{bmatrix}}$$

Feed q back into A.

BFS Walkthrough

$$
\overset{\text{A}}{\begin{bmatrix}
 & & & 1 & & & \\
1 & & 1 & & & & \\
 & & & 1 & & 1 & 1 \\
1 & & & & & & 1 \\
 & 1 & & & & & 1 \\
 & & 1 & & 1 & & \\
 & 1 & & & & &
\end{bmatrix}}
\cdot
\overset{\text{q}}{\begin{bmatrix} 1 \\ \\ 1 \\ \\ \\ \\ \\ \end{bmatrix}}
\rightarrow
\overset{\text{q}}{\begin{bmatrix} \\ 1 \\ \\ 1 \\ \\ 1 \\ \\ \end{bmatrix}}
,
\overset{\text{v}}{\begin{bmatrix} 2 \\ 3 \\ 2 \\ 1 \\ \\ 3 \\ \\ \end{bmatrix}}
$$

Feed q back into A.

Two new nodes are discovered (green), and another was already visited (red)

shgraph

The GraphBLAS spec is a year old, and there are no HPC solutions yet

shgraph is an OpenSHMEM implementation using a light wrapper

Goals:

1. Scale to an arbitrary number of PEs
2. Store sparse graphs of arbitrary size
3. Runtime scales linearly
4. Optimize for memory over time

shgraph is fairly faithful to GraphBLAS

Data Distribution

PEs are logically arranged in a 2D grid

Matrices and vector are divvied up

Assume we have 12 PEs available, and a matrix M and a vector v to store. Each cell below represents a PE:

M_{11}, v_1	M_{12}, v_2	M_{13}, v_3
M_{21}, v_4	M_{22}, v_5	M_{23}, v_6
M_{31}, v_7	M_{32}, v_8	M_{33}, v_9
M_{41}, v_{10}	M_{42}, v_{11}	M_{43}, v_{12}

The more uniformly distributed the matrix data, the better

Data Distribution

Per PE, we use a DCSR matrix representation

- We can do better than just representing matrix members as an array of triples
- Extra-compressed version of CSR

Vectors are distributed to optimize for vector-matrix multiplication

- Assumed to be dense
- nth vector chunk along nth PE row
- Vector is further divided inside each row
- For vxm, the vector components are cycled along the PE row
- The results are then collapsed column-wise

Moral: lots of data is flying around. Matrix-matrix multiplication is worse

Using SHMEM

SPMD is compatible-ish with GraphBLAS

*** Lots of “all-to-all” memory operations

Can’t esaily take advantage SHMEM’s specific type operations

The fact that GraphBLAS objects are opaque has been helpful

Handling runtime errors is hard

Benchmark Results

Comparing against CombBLAS v1.4, a C++ MPI-based sparse graph library that influenced the GraphBLAS standard

Tested on RMAT graphs with the Graph500 parameters [0.57, 0.19, 0.19, 0.05] and edge factor 16

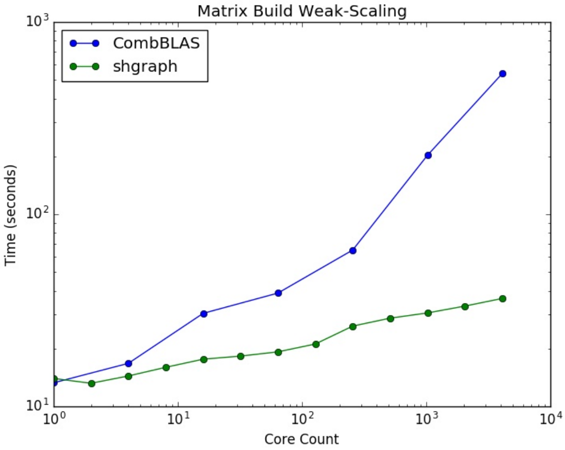
Weak-scaling: graph size is proportional to core count

Two separate benchmarks, one for the matrix build runtime and another for the BFS

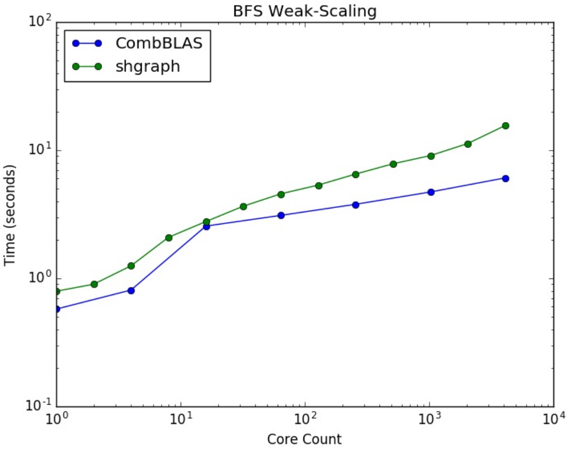
Run on an HPC:

- Cray XC30 system with 128 GB per node
- 2 Intel Haswell CPUs (2.3 GHz sockets) per node
- 16 cores per socket

Matrix Build Results



BFS Results



BFS Weak-Scaling
CombBLAS
shgraph
Time (seconds)
Core Count
10^{2}
10^{1}
10^{0}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}

Next Steps

Finish implementation of the standard

Better computation/communication overlapping

Add multithreading

Still some algorithmic improvements to take advantage of