BLASToff

Graph computation language based on the BLAS* specification

*Basic Linear Algebra Subprograms
Let’s hop right in

01 Motivation
02 Semirings
03 Selection
04 Other features
05 Dirty Details
06 Demo
Motivation

Graphs as matrices

Graphs can be represented as matrices.

Graph operations can be written as matrix operations.

Benefits

Matrix operations are highly optimized, fully realizing parallel computation.
GraphBLAS API
BFS using the C GraphBLAS Library

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "GraphBLAS.h"

/*
 * Given a boolean n x n adjacency matrix A and a source vertex s, performs a BFS traversal
 * of the graph and sets v[i] to the level in which vertex i is visited (v[i] == 1).
 * If i is not reachable from s, then v[i] = 0. (Vector v should be empty on input.)
 */

GrB_Info BFS(GrB_Vector sv, GrB_Matrix A, GrB_Index s) {
    GrB_Index n;
    GrB_Matrix mrows(&n, A);
    // n = # of rows of A
    GrB_Vector v;
    // Vector<int32_t> v(n)
    GrB_Vector q;
    GrB_Vector new(&q, GrB_BOOL, n);
    // Vector<bool> q(n)
    GrB_Vector setElement(&q, (bool)true, s);
    // q[s] = true, false everywhere else
    /*
     * BFS traversal and label the vertices.
     */
    int32_t d = 0;
    bool succ = false;
    do {
        // d = level in BFS traversal
        // succ = true when some successor found
        ++d;
        GrB_multiply(sv, q, GrB_NULL, d, GrB_ALL, n, GrB_NULL); // v[i] = d
        GrB_xor(q, sv, GrB_NONE, GrB_MOD, GrB_BOOL, q, GrB_ALL);
        // q[v] = q | q ⊗ A; finds all the
        // unvisited successors from current q
        GrB_reduce(&q, GrB_NONE, GrB_MOD, GrBBOOL, GrB_NULL, q, GrB_NULL);
        // succ = ||(q)
        // if there is no successor in q, we are done.
        GrB_free(&q);
        // q vector no longer needed
    } while (!succ);
    return GrB_SUCCESS;
}

From “The GraphBLAS C API Specification”, Buluç, et al
BFS in BLASToff

```python
def BFS(G, frontier) {
    #logical;
    N = |G|[0];
    levels = Zero(N : 1);
    maskedGT = G^T;
    depth = 0;
    while (plusColumnReduce(frontier)) {
        #arithmetic;
        depth = depth + 1;
        #logical;
        levels[rangeFromVector(frontier)] = depth;
        mask = !(levels)[0, Zero(N:1), N, 1];
        maskedGT = maskedGT @ mask;
        frontier = maskedGT * frontier;
    }
    #arithmetic;
    return levels + One(|levels|)~(-1);
}
```

There’s a lot going on here. Let’s talk about some of these features!
BLAStoff Overview

- Every object is a matrix
- Imperative
- Wide offering of primitive matrix operations
- Versatile matrix selection operator
- Semiring semantics
What is a semiring?

A set of two binary operators: addition and multiplication.
What is a semiring?

A set of two binary operators: addition and multiplication.

- \((R, +)\) is a commutative monoid with identity element 0
- \((R, \cdot)\) is a monoid with identity element 1
- Multiplication left and right distributes over addition
- Multiplication by 0 annihilates \(R\)
What is a semiring?

A set of two binary operators: addition and multiplication.

Arithmetic semiring:
- $3 + 7 = 10$
- $3 * 7 = 21$
- etc.
What is a semiring?

A set of two binary operators: addition and multiplication.

Arithmetic semiring:
- $3 + 7 = 10$
- $3 \times 7 = 21$
- etc.

Logical semiring:
- $3 + 0 = 1$
- $3 + 7 = 1$
- $0 + 0 = 0$
- $3 \times 0 = 0$
- etc.
A set of two binary operators: addition and multiplication.

What is a semiring?

Arithmetic semiring:
- $3 + 7 = 10$
- $3 \times 7 = 21$
- etc.

Logical semiring:
- $3 + 0 = 1$
- $3 + 7 = 1$
- $0 + 0 = 0$
- $3 \times 0 = 0$
- etc.

Maxmin semiring:
- $3 + 7 = 7$
- $3 \times 7 = 3$
- etc.
Semirings in BLAStoff

#semiring-name; to change semiring
Semirings in BLAS troop

#semiring-name; to change semiring

```
1 A = [ 1, 2;
2 3, 4 ];
3
4 B = [ 0, -1;
5 -2, 5 ];
6
7 #maxmin;
8 printm(A + B);   // prints:  1 2
9 printm(A * B);   // prints: -2 2

11 #arithmetic;
12 printm(A + B);   // prints:  1 1
13 printm(A * B);   // prints: -4 9
```
def addThree(A, B, C) {
    sum = A + B + C;
    return sum;
}

def f(A, B, C) {
    #maxmin;
    printm(addThree(A, B, C));  // prints 6
    printm(A + B + C);          // prints 3
}

A = 1;
B = 2;
C = 3;

printm(A + B + C);  // prints 6
f(A, B, C);
def addThree(A, B, C) {
  #maxmin;
  sum = A + B + C;
  return sum;
}

def f(A, B, C) {
  #maxmin;
  printm(addThree(A, B, C));  // prints 3
  printm(A + B + C);          // prints 3
}

A = 1;
B = 2;
C = 3;

printm(A + B + C);  // prints 6
f(A, B, C);
Semirings in BLAStoff

```python
1 def addThree(A, B, C) {
2     #_;
3     sum = A + B + C;
4     return sum;
5 }
6
7 def f(A, B, C) {
8     #maxmin;
9     printm(addThree(A, B, C));  // prints 3
10     printm(A + B + C);        // prints 3
11 }
12
13 A = 1;
14 B = 2;
15 C = 3;
16
17 printm(A + B + C);  // prints 6
18 f(A, B, C);
```
There is wisdom in the selection of wisdom.

– Dr. Bergen Evans, English professor and TV host
How should selection work?

- Robust
- Expressive
- Powerful
- But **concise**
- In other words, matrix.get(i,j) won't cut it
Our selection operator

- M[A, B, c, d]
- A: row indices, B: column indices
- c, d: size of the submatrices
- A is the only required argument
- B, c, d default to [0], [1], [1], respectively
Example

```plaintext
v = [1; 2; 3; 4];
M = [1, 2, 3;
    4, 5, 6;
    7, 8, 9];

v[2]; // gets [3]
M[2, 1]; // gets [8]

v[[0; 3]]; // gets [1; 4]
M[1, 1, 1, 2]; // gets [5, 6]
```
Example

```plaintext
M = [1, 2, 3;
    4, 5, 6;
    7, 8, 9];

M[[1;0], [0;1], 2, 2];

/* gets:
[4, 5, 5, 6;
  7, 8, 8, 9;
  1, 2, 2, 3;
4, 5, 5, 6]; */
```
Example

\[
M = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix};
\]

\[
M[[1;0], [0;1], 2, 2];
\]

/* gets:
\[
\begin{bmatrix}
4 & 5 & 5 & 6 \\
7 & 8 & 8 & 9 \\
1 & 2 & 2 & 3 \\
4 & 5 & 5 & 6
\end{bmatrix}\]*/
Example

```
26  M = [1, 2, 3;
27     4, 5, 6;
28     7, 8, 9];
29
30  M[0, [0;2], 2, 1] = [-1;-1];
31
32  /* sets M to
33     [-1, 2, -1;
34     -1, 5, -1;
35     7, 8, 9];*/
```
"It's not the operation itself that is the concern, it's the anesthesia."

– Sanjay Gupta, neurosurgeon
A ~ B: slide B across A like so...

...where each windowed view becomes just one entry in the resulting matrix.

Why is this useful for us?
Convolution ~

- Can be used to emulate other typical operators, most notably scalar multiplication.
- BLASStoff has no scalars. To achieve this, we just use a sliding window of size 1x1!

A = [1, 2, 3; 4, 5, 6];
k = 2;
B = A ~ k;
// B is now [2, 4, 6; 8, 10, 12];
For an m x n matrix A, |A| returns a 2 x 1 column vector with values m and n.

For instance, to make an m x n matrix of zeros would simply be:

```plaintext
A = [1, 2, 3; 4, 5, 6];
B = Zero(|A|);
// B is now [0, 0, 0; 0, 0, 0];
```
If we isolate the values into separate variables, we can use selection to replace all values of A!

\[
m = |A|[0]; \\
n = |A|[1]; \\
A[\text{range}(m), \text{range}(n)] = 3;
\]
Row-reductions with either summation or product.

A = [1, 2; 3, 4];
B = +%A; // B is [3; 7]
C = *%A; // C is [2; 12]

(And this works with semirings!!)
Another Feature: Graph Literals

Graphs can be declared just like matrices

// These create equivalent matrices
G = [ 0->1; 2->3; 3->0 ]
M = [ 0,1,0,0;
     0,0,0,1;
     0,0,0,0;
     1,0,0,0 ]
Other basic operators

- Matrix multiplication (*)
- Element-wise multiplication (@) and addition (+)
- Exponent: ^\( (b \mid T) \) where \( b \) is a 1x1 matrix and \( b \geq 0 \)
- Vertical concatenation (,:)
Dirty details;
Are we proud?
What We’re Proud Of

01 Our Process
- Excellent division of labor, everyone specialized while still interacting with all the code
- Github issues for feature tracking

02 Our Project
- Implemented our full LRM, save stretch goals
- Learned linear algebra and abstract algebra

03 Our Code
- Removed SAST while keeping type-checking of int vs float matrices
- Programmatically created function types, definitions, and calls
- Lazy evaluation
- Semiring stack
What We’re Not Proud Of:
Our commit messages
BFS Demo

Questions?

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik.