Analysis: cuSPARSE, cuBLAS, and GraphBLAST

For Sparse Graph Algorithm Development on GPUs

Semiring (Refrencing GraphBLAST)

A semiring is the computation on vertex and edge of the graph. In standard matrix multiplication the semiring used is the `(+, x)` arithmetic semiring. In GraphBLAS, when the semiring is applied to this operation, it represents the transformation that is required to transform the input vector into the output vector. What the `(+, x)` represent are:

- `x`: computation per edge, generates up to `num_edges` intermediate elements
- `+`: computation in the reduction of intermediates back down to a subset of vertices, up to `num_verts` elements

The most frequently used semirings (with their common usage in brackets) are:

- `PlusMultipliesSemiring`: arithmetic semiring (classical linear algebra)
- `LogicalOrAndSemiring`: Boolean semiring (graph connectivity)
- `MinimumPlusSemiring`: tropical min-plus semiring (shortest path)

- `MaximumMultipliesSemiring`: tropical max-times semiring (maximal independent set)

1. cuBLAS: Dense Linear Algebra Primitive

Usage

- Optimized for **dense matrix operations** (GEMM, GEMV) using NVIDIA tensor cores.

Use Case: AI/ML dense tensor operations, not graph analytics

---

2. cuSPARSE: Low-Level Sparse Primitives

Usage

- GPU-accelerated SpMV/SpMM (30-150× faster than CPU)

Limitations:

- No native semiring support

Use Case: Sparse linear algebra in CFD/seismic, not graph abstractions.

---

3. GraphBLAST: Semiring-Optimized Graph Analytics

Architectural Advantages

3.1 Native Semiring Support Implements GraphBLAS standard with algebraic graph algorithms:

3.2 GPU-Specific Optimizations

- **Direction optimization**: Auto-switches push/pull traversal
- **Merge-based load balancing**:

```
if (avg_edges < 4) launch_coalesced_kernel();
else launch_workstealing_kernel();
```

- **Code Example (Concise BFS)**:

```
GrB_vxm(levels, GrB_NULL, GrB_LOR_LAND_SEMIRING_INT32, A, levels, GrB_NULL);
```

- **3.3 Single Source Shortest Path**

```
#include "graphblas/graphblas.hpp"

// Single-source shortest-path on adjacency matrix A from source s
graphblas::Info ssspSimple( Vector<float>*       v,
                            const Matrix<float>* A,
                            Index                s,
                            Descriptor*          desc ) {
  // Get number of vertices
  graphblas::Index A_nrows;
  A->nrows(&A_nrows);

  // Distance vector (v)
  std::vector<graphblas::Index> indices(1, s);
  std::vector<float>  values(1, 0.f);
  v->build(&indices, &values, 1, GrB_NULL);

  // Buffer vector (w)
  graphblas::Vector<float> w(A_nrows);

  // Semiring zero vector (zero)
  graphblas::Vector<float> zero(A_nrows);
  zero.fill(std::numeric_limits<float>::max());

  // Initialize loop variables
  graphblas::Index iter = 1;
  float succ_last = 0.f;
  float succ = 1.f;

  do {
    succ_last = succ;
```

```
    // v = v + v * A^T (do relaxation on distance vector v)
    graphblas::vxm<float, float, float, float>(&w, GrB_NULL, GrB_NULL,
        graphblas::MinimumPlusSemiring<float>(), v, A, desc);
    graphblas::eWiseAdd<float, float, float, float>(v, GrB_NULL, GrB_NULL,
        graphblas::MinimumPlusSemiring<float>(), v, &w, desc);

    // w = v < FLT_MAX (get all reachable vertices)
    graphblas::eWiseMult<float, float, float, float>(&w, GrB_NULL, GrB_NULL,
        graphblas::PlusLessSemiring<float>(), v, &zero, desc);

    // succ = reduce(w) (do reduction on all reachable distances)
    graphblas::reduce<float, float>(&succ, GrB_NULL,
        graphblas::PlusMonoid<float>(), &w, desc);
    iter++;

    // Loop until total reachable distance has converged
  } while (succ_last != succ);

  return GrB_SUCCESS;
}
```

The idea behind GraphBLAS is that four concepts can be used to implement many graph algorithms: vector, matrix, operation and semiring.

Vector

A vector is a subset of vertices of some graph.

Matrix

A matrix is the adjacency matrix of some graph.

Operation

An operation is the memory access pattern common to most graph algorithms (equivalent Ligra terminology is shown in brackets):

- `mxv`: matrix-vector multiply (EdgeMap)
- `vxm`: vector-matrix multiply (EdgeMap)
- `mxm`: matrix-matrix multiply (multi-frontier EdgeMap)
- `eWiseAdd`: elementwise addition (VertexMap)
- `eWiseMult`: elementwise multiplication (VertexMap)

Publications

1. Carl Yang, Aydın Buluç, and John D. Owens. **GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU**. arXiv preprint arXiv:1908.01407 (2019). [arXiv]

2. Carl Yang, Aydın Buluç, and John D. Owens. **Design Principles for Sparse Matrix Multiplication on the GPU**. In *Proceedings of the 24th International European Conference on Parallel and Distributed Computing*, Euro-Par, pages 672-687, August 2018. Distinguished Paper and Best Artifact Award. [DOI | http | slides]

3. Carl Yang, Aydın Buluç, John D. Owens. **Implementing Push-Pull Efficiently in GraphBLAS**. In *Proceedings of the International Conference on Parallel Processing*, ICPP, pages 89:1-89:11, August 2018. [DOI | http | slides]

4. Carl Yang, Yangzihao Wang, and John D. Owens. **Fast Sparse Matrix and Sparse Vector Multiplication Algorithm on the GPU. In Graph Algorithms Building Blocks**, In *Graph Algorithm Building Blocks*, GABB, pages 841–847, May 2015. [DOI | http | slides]

---

4. Critical Comparison

Aspect	cuBLAS	cuSPARSE	GraphBLAST
Target	Dense matrices	Sparse matrices	Sparse graphs
Semiring Support	None	Manual emulation	Native (GraphBLAS)
BFS Lines of Code	150+	80	25
Load Balancing	N/A	Manual	Auto-optimized

While cuSPARSE remains valuable for numerical sparse linear algebra, GraphBLAST's focus on graph algorithm's indicates it to be a better choice for now.

Sources:[^1] cuSPARSE Documentation[^2] NVIDIA cuSPARSE Overview[^3] Matrix Multiplication Benchmark (arXiv:2405.17322)[^4] NVIDIA cuBLAS Documentation[^5]

GraphBLAS Forum Collection[^6] GraphBLAST Performance Analysis[^7] UC Davis GraphBLAST Report[^9] GraphBLAST ACM Paper Summary[^10] SuiteSparse:GraphBLAS Implementation