



# Generalized sparse linear algebra framework for multi-GPU computations

**Egor Orachev, 21.M07-mm**

Scientific supervisor: C.Sc., docent S.V. Grigorev  
Saint Petersburg State University

December 29, 2021

# Introduction

## Sparse linear algebra framework:

- Practical problem solving
- High-performance libraries
- Values' types and functions
- Primitives: matrix, vector, scalar
- Operations: mxm, vxm, mxv, assign, reduce, transpose

Note 1: practical data are sparse

Note 2: practical data are large

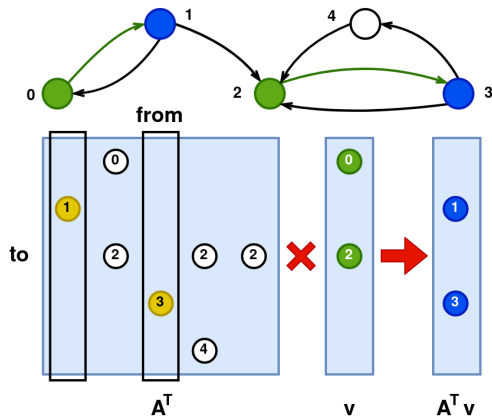


Figure: Graph traversal by matrix-vector multiplication

- **Algorithms**

- ▶ Breadth-first search
- ▶ Shortest paths
- ▶ Maximal independent set
- ▶ Page rank
- ▶ Triangles counting
- ▶ Regular/CFL-reachability

- **Analysis tasks**

- ▶ Static code analysis
- ▶ Graph database queries
- ▶ Bioinformatics

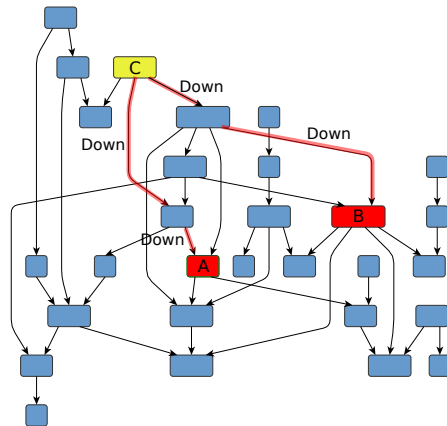


Figure: Navigational query  $\overline{\text{Down}}^n \text{Down}^n$  for CFL-reachability

# Background

- GraphBLAS
  - ▶ Mathematical notation translated into an C API
- GraphBLAS:SuiteSparse
  - ▶ GraphBLAS reference implementation
  - ▶ CPU-computations & high-performance
- GraphBLAST
  - ▶ CUDA C++ template based
  - ▶ Under-development | abandoned
- cuSPARSE, clSPARSE, bhSPARSE, GALATIC, cusp
  - ▶ General-purpose sparse linear algebra libraries
  - ▶ Under-development | outdated | single-GPU
- SPbLA, cuBool
  - ▶ OpenCL | CUDA | CPU
  - ▶ Single-GPU & optimized & boolean values only



## GRAPHBLAS

Figure: GraphBLAS project logo (picture from [graphblas.org](http://graphblas.org))

# Project: motivation and tasks

- **Motivation**

- ▶ No complete and ready for usage GPU GraphBLAS implementation
- ▶ Existing math libraries limited in customization
- ▶ No multi-GPU support

- **Idea**

- ▶ Generalized sparse linear algebra framework
- ▶ Verbose and declarative API
- ▶ No templates  $\Rightarrow$  C and Python wrapping

- **Challenges**

- ▶ GPU programming is hard!
- ▶ Compute APIs verbose and low-level
- ▶ Numerous algorithms for particular operations

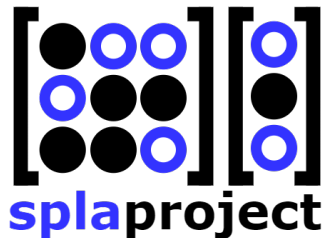


Figure: SPLA project logo  
(picture from spla project page)

# Requirements

- User-defined types support
- User-defined functions support
- DAG-based expressions
- Automated internal hybrid storage format
- Automated multi-GPU work scheduling

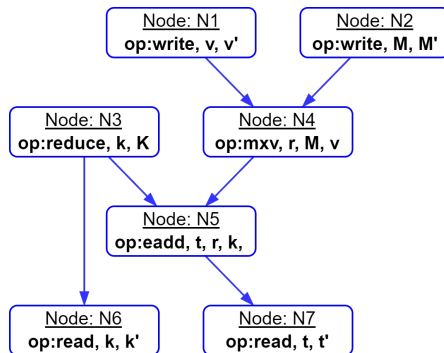


Figure: Example computational expression in DAG form. Note dependencies between nodes.

# Implementation details

- **Dev-stack:**

- ▶ C++17, CMake, C 99, Python 3
- ▶ Compute API: OpenCL 1.2<sup>1</sup>
- ▶ Aux compute library: boost.compute<sup>2</sup>
- ▶ Tasking library: taskflow<sup>3</sup>

- **Strategy:**

- ▶ Write generalized cl kernels
- ▶ Utilize boost meta-kernels library
- ▶ Handle values as raw byte sequences (POD)
- ▶ User-defined functions effectively are strings

<sup>1</sup><https://www.khronos.org/opencl/>

<sup>2</sup><https://github.com/boostorg/compute>

<sup>3</sup><https://github.com/taskflow/taskflow#dynamic-tasking>

```
template<class InputIterator, class MapIterator, class OutputIterator>
class gather_kernel : public meta_kernel
{
public:
    gather_kernel() : meta_kernel("gather")
    {}

    void set_range(MapIterator first,
                  MapIterator last,
                  InputIterator input,
                  OutputIterator result)
    {
        m_count = iterator_range_size(first, last);

        *this <<
            "const uint i = get_global_id(0);\n" <<
            result[expr<uint>("i")] << "=" <<
            input[first[expr<uint>("i")]] << "; \n";
    }

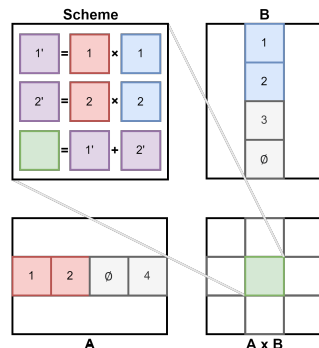
    event exec(command_queue &queue)
    {
        if(m_count == 0) {
            return event();
        }

        return exec_id(queue, 0, m_count);
    }

private:
    size_t m_count;
};
```

Figure: Gather OpenCL meta-kernel (picture from boost.compute library)

- Work distribution between computational units
- Host-device synchronization
- Computational dependencies
- Storage format and data sharing
- Efficient CPU usage



**Figure:** Example blocked sparse matrix storage. Evaluation of  $A \times B$  for some result block. Note, each temporary block product is fully independent GPU task.



# API Usage example

```
1  /* !!! Want to evaluate expression `v = b x A + reduce(K)` */
2
3  /* 1. Make nodes */
4  spla_Expression_MakeVxM(e, &node_bxA, tmp1, NULL, mult, plus, b, A, NULL);
5  spla_Expression_MakeMatrixReduce(e, &node_rK, tmp2, NULL, plus, K, NULL);
6  spla_Expression_MakeVectorEAdd(e, &node_eadd, v, NULL, plus, tmp1, tmp2, NULL);
7
8  /* 2. Setup deps */
9  spla_Expression_Dependency(e, node_bxA, node_eadd);
10 spla_Expression_Dependency(e, node_rK, node_eadd);
11
12 /* 3. Submit */
13 spla_Expression_Submit(e);
14
15 /* 4.1 Do some usefull stuff and return back when ready */
16 spla_Expression_GetState(e, &state);
17
18 /* 4.2 Or block until ready */
19 spla_Expression_Wait(e);
```

**Figure:** Evaluation of  $v = b \times A + \text{reduce}(K)$  using **spla C API**, where  $v$ ,  $b$  are vectors,  $A$  and  $K$  are matrices, defined somewhere earlier in the code. DAG is set up using expression node dependencies. Note, that we use generic *plus* and *mult*, which can be defined by the user.

- C++ Core API
- Primitives: matrix, vector, scalar
- Operations: mxm, vxm, assign, eadd
- Algorithms: *bfs*, *sssp*
- Hybrid storage format
  - ▶ Generic blocked matrix, vector
  - ▶ COO-format blocks
  - ▶ Empty blocks not stored
  - ▶ It is possible to add CSR, DCSR, Dense blocks
- Task-graph
  - ▶ Task for each DAG's node
  - ▶ Separate set of sub-tasks inside each node's task
  - ▶ Each sub-task is assigned to concrete GPU

- Operations: reduce, emult, transpose
- Algorithms: *tc*, *page-rank*, *connected-components*, etc.
- Fine tuning: optimizations, state of the art SpGEMM's
- C API wrapper & Python package (PyPI publication)

- SPLA project: <https://github.com/JetBrains-Research/spla>
- Email: [egororachyov@gmail.com](mailto:egororachyov@gmail.com)
- Materials:
  - ▶ Szuppe, J. 2016. Boost.Compute: A parallel computing library for C++ based on OpenCL. Proceedings of the 4th International Workshop on OpenCL.
  - ▶ Timothy A. Davis. 2019. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. ACM Trans. Math. Softw. 45, 4, Article 44 (December 2019), 25 pages. DOI:<https://doi.org/10.1145/3322125>
  - ▶ E. Orachev, M. Karpenko, A. Khoroshev and S. Grigorev. 2021. "SPbLA: The Library of GPGPU-Powered Sparse Boolean Linear Algebra Operations," IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2021, pp. 272-275, doi: 10.1109/IPDPSW52791.2021.00049.