Generalized sparse linear algebra framework for GPU computations

Egor Orachev, 21.M07-мм

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

June 3, 2022

Introduction

Sparse linear algebra framework:

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

Note 1: practical data is sparse
Note 2: practical data is large

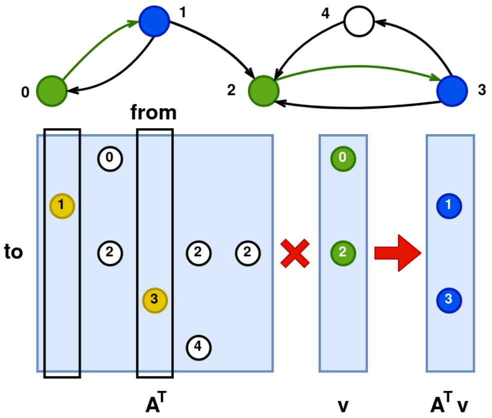


Figure: Graph traversal by matrix-vector multiplication

Applications

- **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

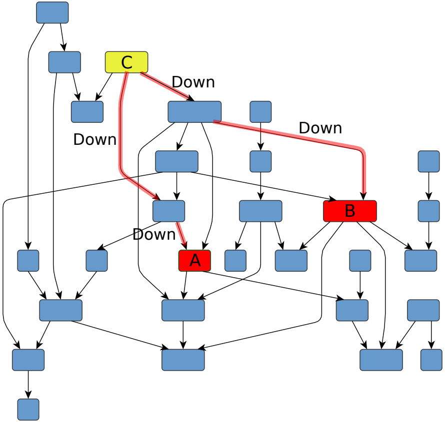


Figure: Navigational query $\overline{\textbf{Down}}^n\,\textbf{Down}^n$ for CFL-reachability

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  - Under-development | outdated | single-GPU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

Figure: GraphBLAS project logo (picture from graphblas.org)

GPU programming challenges

- Complex APIs
- Different algorithms
- Workload imbalance
- Irregular access patterns
- Fine-grained parallelism
- Minimizing overhead
- Computations intensity

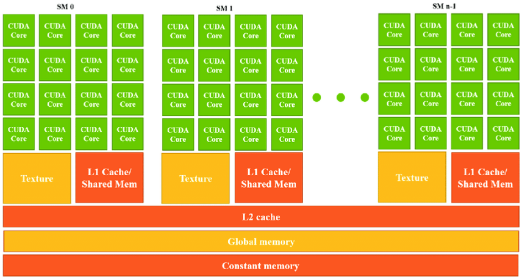


Figure: Schematic of NVIDIA GPU architecture

Project: motivation and tasks

- **Motivation**
  - No complete and ready for usage GPU GraphBLAS implementation
  - Existing math libraries limited in customization
- **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)

Problem statement

The goal of this work is the implementation of the generalized sparse linear algebra primitives and operations library for GPU computations.

Tasks:

- Develop the architecture of the library
- Implement the library accordingly to the developed architectur
- Implement a set of most common graph algorithms using library
- Conduct experimental study of implemented artifacts

Project requirements

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

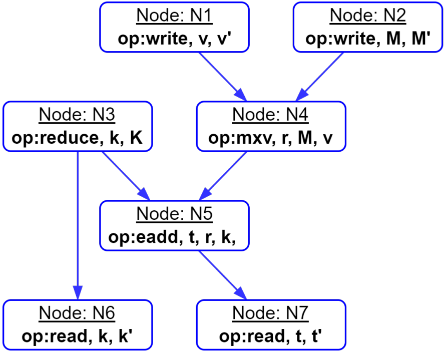
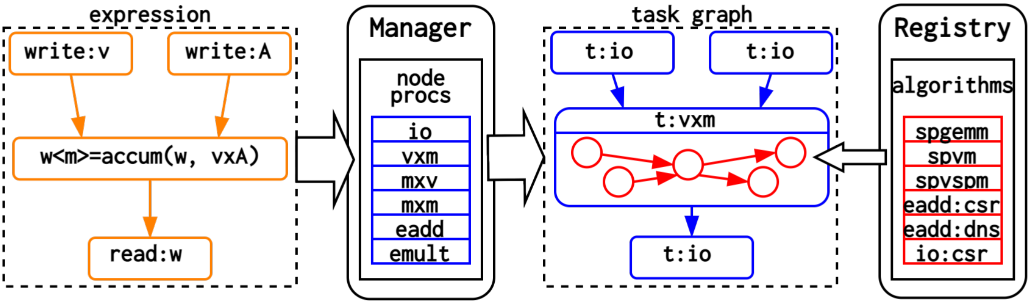
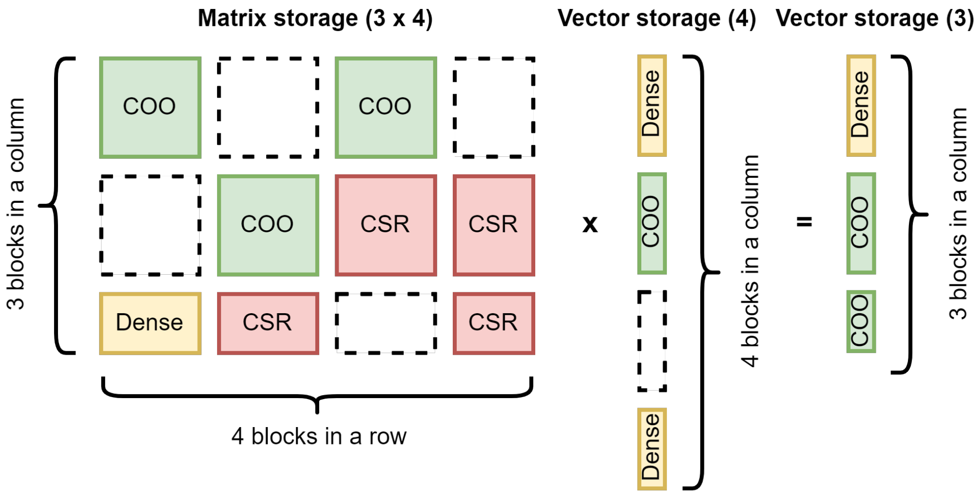


Figure: Example computational expression in DAG form. Note dependencies between nodes

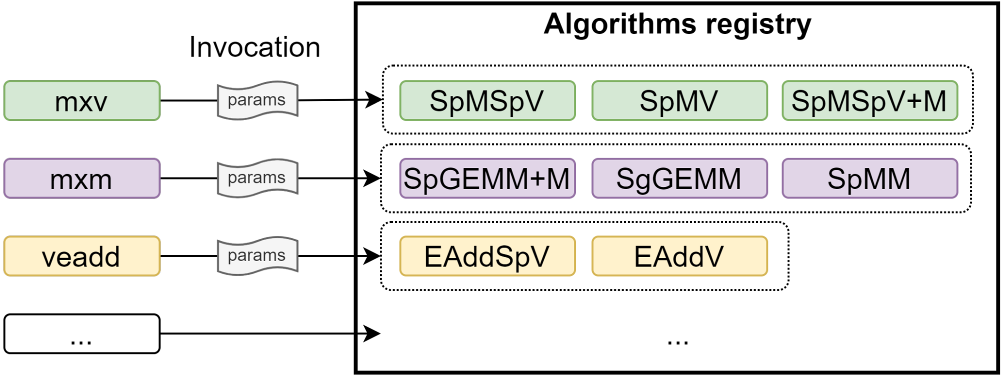
Execution model



Blocked storage based matrix-vector product



Idea of algorithm invocation segregation



Implementation details

- **Dev-stack**:
  - C++17, CMake, C 99, Python 3
  - Compute API: OpenCL 1.2[1]
  - Aux compute library: boost.compute[2]
  - Tasking library: taskflow[3]
- **Strategy**:
  - Write generalized cl kernels
  - Utilize boost meta-kernels library
  - Handle values as raw byte sequences (POD)
  - User-defined functions effectively are strings

```cpp
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_1d(queue, 0, m_count);
    }

private:
    size_t m_count;
};
```

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

[1] https://www.khronos.org/opencl/
[2] https://github.com/boostorg/compute
[3] https://github.com/taskflow/taskflow#dynamic-tasking

Breadth-first search algorithm implementation using Spla API

```
while (sp_q→GetNvals() != 0) {
    auto sp_iter = Expression::Make(library);
    auto t1 = sp_iter→MakeDataWrite(sp_depth, DataScalar::Make(&depth, library));
    auto t2 = sp_iter→MakeAssign(sp_v, sp_q, nullptr, sp_depth, sp_desc_accum);
    auto t3 = sp_iter→MakeVxM(sp_q, sp_v, nullptr, nullptr, sp_q, sp_A, sp_desc_comp);

    if (!sparseToDense && sp_v→GetFillFactor() ≥ denseFactor) {
        auto tt = sp_iter→MakeToDense(sp_v, sp_v);
        sp_iter→Dependency(tt, t2);
        sparseToDense = true;
    }

    sp_iter→Dependency(t1, t2);
    sp_iter→Dependency(t2, t3);
    sp_iter→SubmitWait();

    depth += 1;
}
```

Experimental study

- Research questions:
  - **RQ1**. What is the performance of the proposed solution relative to existing tools for both CPU and GPU analysis?
  - **RQ2**. What is the portability of the proposed solution with respect to various device vendors and OpenCL runtimes?
- Setup:
  - PC with Ubuntu 20.04
  - 3.40Hz Intel Core i7-6700 4-core CPU
  - DDR4 64Gb RAM
  - Intel HD Graphics 530 integrated GPU
  - Nvidia GeForce GTX 1070 dedicated GPU, 8Gb on-board VRAM

Dataset description

Dataset	Vertices	Edges	Max Degree
coAuthorsCiteseer	227.3K	1.6M	1372
coPapersDBLP	540.4K	30.4M	3299
hollywood-2009	1.1M	113.8M	11,467
roadNet-CA	1.9M	5.5M	12
com-Orkut	3M	234M	33313
cit-Patents	3.7M	16.5M	793
rgg_n_2_22_s0	4.1M	60.7M	36
soc-LiveJournal	4.8M	68.9M	20,333
indochina-2004	7.5M	194.1M	256,425

Figure: Dataset. Matrices were selected from the Sparse Matrix Collection at University of Florida. All datasets are converted to undirected graphs. Self-loops and duplicated edges are removed

BFS results

Dataset	Nvidia			Intel	
	GR	GB	SP	SS	SP
hollywood-2009	20.3	82.3	36.9	23.7	303.4
roadNet-CA	33.4	130.8	1456.4	168.2	965.6
soc-LiveJournal	60.9	80.6	90.6	75.2	1206.3
rgg_n_2_22_s0	98.7	414.9	4504.3	1215.7	15630.1
com-Orkut	205.2	− −	117.9	43.2	903.6
indochina-2004	32.7	− −	199.6	227.1	2704.6

Figure: Breadth-first search algorithm evaluation results. Time in milliseconds. Tools: Gunrock (GR), GraphBLAST (GB), SuiteSparse (SS), Spla (SP)

BFS results

Dataset	Nvidia			Intel	
	GR	GB	SP	SS	SP
hollywood-2009	20.3	82.3	36.9	23.7	303.4
roadNet-CA	33.4	130.8	1456.4	168.2	965.6
soc-LiveJournal	60.9	80.6	90.6	75.2	1206.3
rgg_n_2_22_s0	98.7	414.9	4504.3	1215.7	15630.1
com-Orkut	205.2	– –	117.9	43.2	903.6
indochina-2004	32.7	– –	199.6	227.1	2704.6

Figure: Breadth-first search algorithm evaluation results. Time in milliseconds. Tools: Gunrock (GR), GraphBLAST (GB), SuiteSparse (SS), Spla (SP)

BFS results

Dataset	Nvidia			Intel	
	GR	GB	SP	SS	SP
hollywood-2009	20.3	82.3	36.9	23.7	303.4
roadNet-CA	33.4	130.8	1456.4	168.2	965.6
soc-LiveJournal	60.9	80.6	90.6	75.2	1206.3
rgg_n_2_22_s0	98.7	414.9	4504.3	1215.7	15630.1
com-Orkut	205.2	– –	117.9	43.2	903.6
indochina-2004	32.7	– –	199.6	227.1	2704.6

Figure: Breadth-first search algorithm evaluation results. Time in milliseconds. Tools: Gunrock (GR), GraphBLAST (GB), SuiteSparse (SS), Spla (SP)

TC results

Dataset	Nvidia			Intel	
	GR	GB	SP	SS	SP
coAuthorsCiteseer	2.1	2.0	9.5	17.5	64.9
coPapersDBLP	5.7	94.4	201.9	543.1	1537.8
roadNet-CA	34.3	5.8	16.1	47.1	357.6
com-Orkut	218.1	1583.8	2407.4	23731.4	15049.5
cit-Patents	49.7	52.9	90.6	698.3	684.1
soc-LiveJournal	69.1	449.6	673.9	4002.6	3823.9

Figure: Triangles counting algorithm evaluation results. Tools: Gunrock (GR), GraphBLAST (GB), SuiteSparse (SS), Spla (SP)

TC results

Dataset	Nvidia			Intel	
	GR	GB	SP	SS	SP
coAuthorsCiteseer	2.1	2.0	9.5	17.5	64.9
coPapersDBLP	5.7	94.4	201.9	543.1	1537.8
roadNet-CA	34.3	5.8	16.1	47.1	357.6
com-Orkut	218.1	1583.8	2407.4	23731.4	15049.5
cit-Patents	49.7	52.9	90.6	698.3	684.1
soc-LiveJournal	69.1	449.6	673.9	4002.6	3823.9

Figure: Triangles counting algorithm evaluation results. Tools: Gunrock (GR), GraphBLAST (GB), SuiteSparse (SS), Spla (SP)

TC results

Dataset	Nvidia			Intel	
	GR	GB	SP	SS	SP
coAuthorsCiteseer	2.1	2.0	9.5	17.5	64.9
coPapersDBLP	5.7	94.4	201.9	543.1	1537.8
roadNet-CA	34.3	5.8	16.1	47.1	357.6
com-Orkut	218.1	1583.8	2407.4	23731.4	15049.5
cit-Patents	49.7	52.9	90.6	698.3	684.1
soc-LiveJournal	69.1	449.6	673.9	4002.6	3823.9

Figure: Triangles counting algorithm evaluation results. Tools: Gunrock (GR), GraphBLAST (GB), SuiteSparse (SS), Spla (SP)

Results

- The architecture of the library for a generalized sparse linear algebra for GPU computations was developed
- The implementation of the library accordingly to the developed architecture was started
- Several algorithms for a graph analysis were implemented using developed library API
- Preliminary experimental study of the proposed artifacts was conducted

Tasks to be done

- Extend a set of available linear algebra operations
- Implement a set of a common graph analysis algorithms: *page-rank*, *connected-components*, *sssp*, etc.
- Conduct a complete experimental study of the set of common graph analysis algorithms

Also

- SPLA project: https://github.com/JetBrains-Research/spla
- Email: 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.