Generalized sparse linear algebra framework for multi-GPU computations

Egor Orachev, 21.M07-мм

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

December 29, 2021

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

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

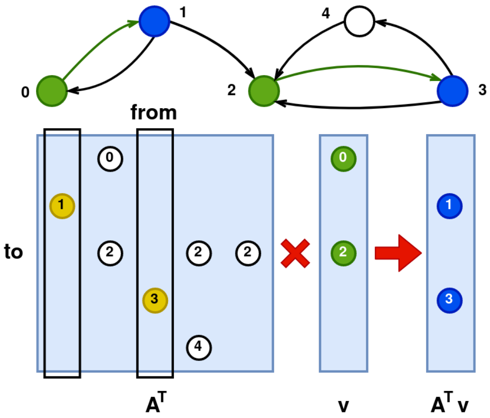

Figure: Graph traversal by matrix-vector multiplication

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

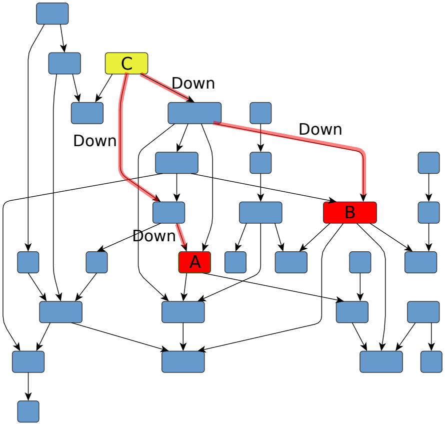

Figure: Navigational query $\overline{\textbf{Down}}^n\textbf{Down}^n$ for CFL-reachability

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  - Under-development | outdated | single-GPU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)

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

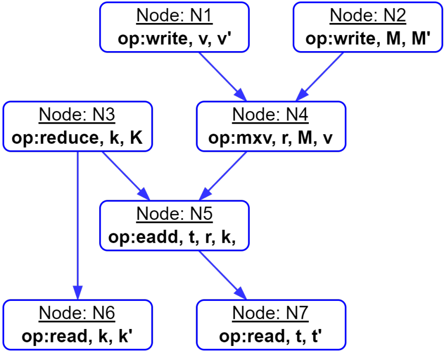

Figure: Example computational expression in DAG form. Note dependencies between nodes.

Implementation details

- **Dev-stack**:
  - C++17, CMake, C 99, Python 3
  - Compute API: OpenCL 1.2[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

Issues

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

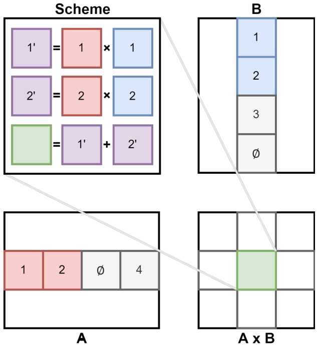

Figure: Example blocked sparse matrix storage. Evaluation of $A \times B$ for some result block. Note, each temporary block product is fully independent GPU task.

API Usage example

```c
/* !!! Want to evaluate expression `v = b x A + reduce(K)`  */

/* 1. Make nodes */
spla_Expression_MakeVxM(e, &node_bxA, tmp1, NULL, mult, plus, b, A, NULL);
spla_Expression_MakeMatrixReduce(e, &node_rK, tmp2, NULL, plus, K, NULL);
spla_Expression_MakeVectorEAdd(e, &node_eadd, v, NULL, plus, tmp1, tmp2, NULL);

/* 2. Setup deps */
spla_Expression_Dependency(e, node_bxA, node_eadd);
spla_Expression_Dependency(e, node_rK, node_eadd);

/* 3. Submit */
spla_Expression_Submit(e);

/* 4.1 Do some usefull stuff and return back when ready */
spla_Expression_GetState(e, &state);

/* 4.2 Or block until ready */
spla_Expression_Wait(e);
```

Figure: Evaluation of $v = b \times A + reduce(K)$ 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.

Progress

- 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

Roadmap

- 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)

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.