Sparse Boolean Linear Algebra on GPGPU*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Abstract is very abstract. Abstract is very abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

One of the techniques to effectively solve a data analysis problem is to reduce it to linear algebra operations over vectors and matrices for appropriate values set. That gives one well studied for years mathematical apparatus, as well as the possibility to evaluate this problem with *zero-cost* by linear algebra libraries, which utilize modern hardware, provide various optimization techniques and allow quickly and safely prototype solution in code with predefined building blocks.

Particularly, in graph data analysis such reduction is well presented, since a graph could be converted to the matrix and with specially defined *semiring* it could be effectively process in linear algebra fashion. Examples of problems solved in this way are all-pairs shortest path, breadth-first search, maximal independent set problems [**?**]. Since practical data often come with huge size and sparse form, what is also applicable for graphs, it requires special processing tools for analysis, what appeals to sparse linear algebra libraries.

Huge amount data analysis typically processed as small chunks with the fixed or rarely changed set of instructions, what relates to the *single instruction, multiple data* (SIMD) model. Although modern CPUs exploit SIMD optimizations, GPGPU gives much more power in such kind data processing at cost of implementation challenges. GPGPU programming introduces heterogeneous device model into system, memory traffic and data operations limitations, as well as requires to take into account vendor-specific capabilities.

At this moment there are presented modern open-source and proprietary sparse linear algebra libraries on GPGPU for general operations and common types of values. However, sparse boolean linear algebra on GPGPU is still not presented, because of its high specificity. Boolean algebra allows to address problems over finite set of values, for example, reachability or relational queries for some graphs [**?**].

In this work we present the sparse boolean linear algebra operations implementation as sand-alone self-sufficient programming libraries for the two most popular GPGPU platforms: NVIDIA Cuda and OpenCL. Cuda is a GPGPU technology for NVIDIA devices, which allows to employ some platform-specific facilities, such as unified memory mechanism, and make an architectural assumptions, what gives more optimizations space at cost of portability. OpenCL is platform agnostic API standard, which allows to run computations on different platforms, such as multi-threaded CPUs, GPUs, and FPGA. Our implementation relies on modern sparse matrices processing techniques, as well as exploits some optimizations, related to the boolean data processing.

II. Related work

Existing libraries, algorithns, frameworks.

III. Implementation Details

Implemented sparse boolean linear algebra libraries for OpenCL and NVIDIA Cuda platforms are called *clBool* and *cuBool* respectively. Source code and related artefacts are available at GitHub hosting [**?**], [**?**]. Libraries are written in the C++ programming language, which is well-suited performance and resource critical computational tasks. Libraries expose C compatible API with relatively small amount of functions and

Identify applicable funding agency here. If none, delete this.

primitives, what gives expressiveness and allows to embed that API into other execution environments via interoperability mechanisms, for example, into Python or .NET runtimes. Libraries source code compilation is configured with CMake tool. Compilation process is straightforward and requires setup of only basic components and instruments, such as compiler, build configuration tools and platform-specific development kits.

Libraries operate on boolean semiring with values set {*true*, *false*} with *false* as a neutral element, '+' operation defined as logical *or* and '*' defined as logical *and*. Values are also denoted as $\{1,\ 0\}$ respectively, and the abbreviation *NNZ(M)* gives the number of non-zero cells of the matrix M.

Main primitive is sparse matrix of boolean values, stored in one of the sparse formats. Sparse vector primitive is not presented, since its utilization is relatively rare presented in practical computational tasks. But its support is something to be added in far future. Primary available operations and functions are following.

- Create sparse matrix M of size $m \times n$.
- Delete sparse matrix M and release all its internal resources.
- Fill the matrix M with values $L = \{(i, j)_k\}_k$. The result of this operation is $M_{i,j} =$ *true* for each $(i, j) \in L$, and $M_{i,j} =$ *false* for the rest of matrix values.
- Read matrix M values $L = \{(i, j) \mid M_{i,j} =$ *true*$\}$.
- Matrix-matrix multiply-add operation $C \mathrel{+}= M \times N$.
- Matrix-matrix add operation $M \mathrel{+}= N$.
- Matrix-matrix Kronecker product $K = M \otimes N$.

A. cuBool

cuBool is sparse boolean linear algebra implementation specifically for NVIDIA Cuda platform. Core of this library relies on Cuda C/C++ language and API, what with NVCC compiler allows intermix C++ with Cuda specifics. Also cuBool employs NVIDIA Thrust auxiliary library, which provides implementation for generic data containers and operations, such as *iterating*, *exclusive or inclusive scan*, *map* and etc., which are executed on Cuda device. That allows express algorithms in terms of high-level optimised primitives, what increases code readability and reduces time for development.

Sparse matrix primitive is stored in the *compressed sparse row* (CSR) format with only two arrays: $rowspt$ for row offset indices and $cols$ for columns indices. Boolean matrices has no actual values, thus $true$ values are encoded only as (i, j) pairs. It allows to store matrix M of size $m \times n$ in $(m +$ *NNZ(M)*$) \times$ *sizeof(IndexType)* bytes of GPU memory, where *IndexType* is type of stored indices, for simplicity can be selected as *uint32_t*.

The algorithm Nspasrse [1] is used for matrix-matrix multiplication. This algorithm is a boolean values case adaptation of the state-of-the-art, efficient and memory saving sparse general matrix multiplication (SpGEMM) algorithm, proposed in Yusuke Nagasaka et al. research [2]. This algorithm was selected because it gives promising relatively small memory footprint for large matrices processing, as well as it competes with other major Cuda SpGEMM implementations, such as cuSPARSE or CUSP.

Matrix-matrix addition is based on GPU Merge Path algorithm [3] with dynamic work balancing and two pass processing. These optimizations give better workload dispatch among execution blocks and allow more precise memory allocations in order to keep memory footprint small respectively.

As an example of library C API embedding, cuBool provides python wrapper, called Pycubool. This module exports library functionality via default CTypes module for native functions calling and provides safe and automated management for native resources.

B. clBool

clBool is sparse boolean linear algebra implementation for OpenCL platform. This library is implemented in the C++ with OpenCL kernels, stored as separate source files, loaded on demand at runtime.

Sparse matrix primitive is stored in *coordinate format* (COO) with two arrays: $rows$ and $cols$ for row and column indices of the stored non-zero values. For the matrix M of size $m \times n$ memory consumption is $2 \times$*NNZ(M)*$\times$*sizeof(IndexType)*. This format was selected instead of CSR, because COO gives better memory footprint for very sparse matrices with a lot of empty rows.

!!! Matrix-matrix multiplication !!!

Matrix-matrix addition is based on GPU Merge Path algorithm as well. Since all COO matrix values are stored in the single array, its merge can be completed at single time, compared to CSR matrix merge computed on a per row basis. This operation is implemented in a classic one pass fashion: it allocates single merge buffer of size *NNZ(A)*+*NNZ(B)*) before actual merge of matrices A and B, what can negatively affect memory consumption for large matrices with lots of duplicated non-zero values at the same positions.

!!! Something about managed environment wrapper !!!

IV. Evaluation

Evaluation of the proposed implemenation(s).

V. Conclusion

Conclusion and future work.

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

[1] A. Terekhov, A. Khoroshev, R. Azimov, and S. Grigorev, "Context-free path querying with single-path semantics by matrix multiplication," 06 2020, pp. 1–12.

[2] Y. Nagasaka, A. Nukada, and S. Matsuoka, "High-performance and memory-saving sparse general matrix-matrix multiplication for nvidia pascal gpu," in *2017 46th International Conference on Parallel Processing (ICPP)*, 2017, pp. 101–110.

[3] O. Green, R. Mccoll, and D. Bader, "Gpu merge path: a gpu merging algorithm," 11 2014.