Spla: Open-Source Generalized Sparse Linear Algebra Framework with Vendor-Agnostic GPUs Accelerated Computations

Anonymous Author(s)

ABSTRACT

Scalable high-performance graph analysis is an actual nontrivial challenge. Usage of sparse linear algebra operations as building blocks for graph analysis algorithms, which is a core idea of GraphBLAS standard, is a promising way to attack it. While it is known that sparse linear algebra operations can be efficiently implemented on GPGPU, full GraphBLAS implementation on GPGPU is a nontrivial task that is almost solved by GraphBLAST project. Though it is shown that utilization of GPGPUs for GraphBLAS implementation significantly improves performance, portability and scalability problems are not solved yet: GraphBLAST uses Nvidia stack and utilizes only one GPGPU. In this work we propose a Spla library that aimed to solve these problems: it uses OpenCL to be portable and designed to utilize multiple GPGPUs. Preliminary evaluation shows that while further optimizations are required, the proposed solution demonstrates performance comparable with GraphBLAST on some tasks. Moreover, our solution on embedded GPU outperforms SuiteSparse:GrpaphBLAS on the respective CPU on some graph analysis tasks.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

graphs, algorithms, graph analysis, sparse linear algebra, GraphBLAS, GPGPU, OpenCL

ACM Reference Format:

Anonymous Author(s). 2018. Spla: Open-Source Generalized Sparse Linear Algebra Framework with Vendor-Agnostic GPUs Accelerated Computations. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Scalable high-performance graph analysis is an actual challenge. There is a big number of ways to attack this challenge [1] and the first promising idea is to utilize general-purpose graphic processing units (GPGPU). Such existing solutions, as CuSha [5] and Gunrock [6] show that utilization of GPUs can improve the performance of graph analysis, moreover it is shown that solutions may be scaled to multi-GPU systems. But low flexibility and high complexity of API are problems of these solutions.

The second promising thing which provides a user-friendly API for high-performance graph analysis algorithms creation is a GraphBLAS API [4] which provides linear algebra based building blocks to create graph analysis algorithms. The idea of GraphBLAS is based on a well-known fact that linear algebra operations can be efficiently implemented on parallel hardware. Along with that, a graph can be natively represented using matrices: adjacency matrix, incidence matrix, etc. While reference CPU-based implementation of GraphBLAS, SuiteSparse:GraphBLAS [2], demonstrates good performance in real-world tasks, GPU-based implementation is challenging.

One of the challenges in this way is that real data are often sparse, thus underlying matrices and vectors are also sparse, and, as a result, classical dense data structures and respective algorithms are inefficient. So, it is necessary to use advanced data structures and procedures to implement sparse linear algebra, but the efficient implementation of them on GPU is hard due to the irregularity of workload and data access patterns. Though such well-known libraries as cuSPARSE show that sparse linear algebra operations can be efficiently implemented for GPGPU, it is not so trivial to implement GraphBLAS on GPGPU. First of all, it requires *generic* sparse linear algebra, thus it is impossible just to reuse existing libraries which are almost all specified for operations over floats. The second problem is specific optimizations, such as masking fusion, which can not be natively implemented on top of existing kernels. Nevertheless, there is a number of implementations of GraphBLAS on GPGPU, such as GraphBLAST [8], GBTL [9], which show that GPGPUs utilization can improve the performance of GraphBLAS-based graph analysis solutions. But these solutions are not portable because they are based on Nvidia Cuda stack. Moreover, the scalability problem is not solved: all these solutions support only single-GPU, not multi-GPU computations.

To provide portable GPU implementation of GraphBLAS API we developed a *SPLA* library[1]. This library utilizes OpenCL for GPGPU computing to be portable across devices of different vendors. Moreover, it is initially designed to utilize multiple GPGPUs to be scalable. To sum up, the contribution of this work is the following.

- Design of portable GPU GraphBLAS implementation proposed. The design involves the utilization of multiple GPUS. Additionally, the proposed design is aimed to simplify library tuning and wrappers for different high-level platforms and languages creation.
- Subset of GraphBLAS API, including such operations as masking, matrix-matrix multiplication, matrix-matrix e-wise addition, is implemented. The current implementation is limited by COO and CSR matrix representation format and uses basic algorithms for some operations, but work

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1] Source code available at: https://github.com/JetBrains-Research/spla

in progress and more data formats will be supported and advanced algorithms will be implemented in the future.
- Preliminary evaluation on such algorithms as breadth-first search (BFS) and triangles counting (TC), and real-world graphs shows portability across different vendors and promising performance: for some problems Spla is comparable with GraphBLAST. Surprisingly, for some problems, the proposed solution on embedded Intel graphic card shows better performance than SuiteSparse:GraphBLAS on the respective CPU. At the same time, the evaluation shows that further optimization is required.

2 BACKGROUND OF STUDY

2.1 Related Work

Related work, existing solutions, systems.

2.2 GraphBLAS

GrpahBLAS API. Discussion of drawbacks of current design and implementation.

2.3 GPU computations

Technologies, problems, challenges, architectures.

3 PROPOSED SOLUTION DESCRIPTION

This section describes the high-level details of the proposed solution. It highlights the design principles, high-level architecture of the solution, data storage representation, operations, and also shows differences from the GraphBLAS API.

3.1 Design Principles

Spla library is designed the way to maximize potential library performance, simplify its implementation and extensions, and to provided the end-user verbose, but effective interface allowing customization and precise control over operations execution. These ideas are captured in the following principles.

- *Optional acceleration.* Library is designed in a way, that GPU acceleration is fully optional part. Library can perform computations using standard CPU pipeline. If GPU is supported, library can offload a part of a work for accelerator. Multiple acceleration backend can be presented in the system.
- *User-defined functions.* The user can create custom element-by-element functions to parameterize operations. Custom functions can be used for both CPU and GPU execution.
- *Predefined scalar data types.* The library provides a set of built-in scalar data types that have a natural one-to-one relationship with native GPU built-in types. Data storage is transparent. The library interprets the data as POD-structures. The user can interpret individual elements as a sequence of bytes of a fixed size.
- *Hybrid-storage format.* The library automates the process of data storage and preprocessing. It supports several data formats, chooses the best one depending on the situation.
- *Exportable interface.* The library has a C++ interface with an automated reference-counting and with no-templates usage. It can be wrapped by C99 compatible API and exported to other languages, for example, in a form of a Python package.

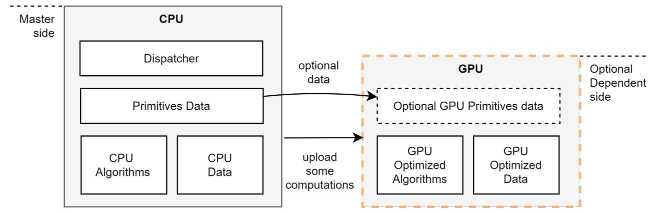

Figure 1: Proposed solution general design idea.

3.2 Architecture Overview

The general idea of the proposed solution is depicted in Fig. 1. The core of the library and its main part is the CPU, which is the master node and controls all calculations. It is responsible for storing data, maintaining a registry with algorithms, and scheduling operations to perform. In this paradigm, the GPU is an optional backend for acceleration, implemented through a special interface. It can optionally store data in a specific format. The CPU can offload the calculation of a part of the operations to the GPU, if the corresponding operation is supported by the given accelerator.

The reason for this is that the CPU and GPU are inherently asymmetric in nature. The GPU is a purely dependent device that must be controlled from the outside. In addition, access to data on the GPU and their storage is carried out differently due to the peculiarities of the execution of kernels. Also, video memory is many times more expensive and it is available many times less. Therefore, RAM is a cache for VRAM, and data duplication can be neglected. In the end, the explicit separation of the CPU model from the GPU of the backend gives modularity. This can be used not only to support different GPU technologies, but also to integrate multiple GPUs or distributed processing in the future.

3.3 Data Containers

Library provides general *M-by-N Matrix*, *N Vector* and *Scalar* data containers. Underlying primitive scalar types are specified by *Type* object. Single vertex or matrix is stored in specialized storage container. An example of the single vector storage is depicted in Fig. 2.

The storage is responsible for keeping data in multiple different formats at the same time. Each format is best suited for a specific type of task and requested on demand. Frequent insertion or deletion requires key-value storage. Mathematical operations use sparse or dense formats. GPU operations require separate format with a copy of the data resident in video memory.

Data transformation from one format to another is carried out using a special rules graph shown in Fig. 3. The directed edges in this graph indicate the conversion rule. The graph must be connected, and any vertex is reachable. An example of the data transformation process is depicted in Fig. 4. If no valid the it is initialized as empty. Otherwise, for a requested format the best path of convertation is obtained. Currently, the shortest on is used. Weight assignment to rules can potentially be used to prioritize convertation and reduce the cost for some formats.

Currently, several storage formats are supported. There is dictionary of keys for vector and matrix (DoK), list of coordinates (COO), dense vector, list of lists (LIL) and compressed sparse rows (CSR) matrix formats. Other formats, such as CSC, DCSR, ELL, etc., can be added to the library by the implementation of formats conversion and by the specialization of operations for a specific format.

3.4 Algebraic Operations

Library provides a number of commonly used operations, such as *vxm*, *mxv*, *mxmT*, *element-wise add*, *assign*, *map*, *reduce*, etc. Other operations can be added on demand. Interface of operations is inspired by GraphBLAS standard. It supports *masking*, parametrization by *binary mult* and *binary add* functions, *select* for filtering and mask application, *unary op* for values transformation, and *descriptor* object for additional operation tweaking.

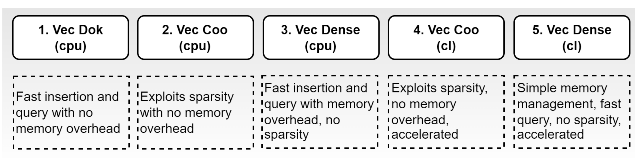

Figure 2: Vector primitive storage. Stores the same data potentially in multiple different formats. Some slots can be empty.

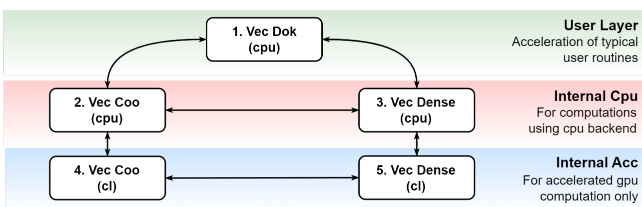

Figure 3: Vector storage transformation graph. The graph defines how data can be obtained from one format in another.

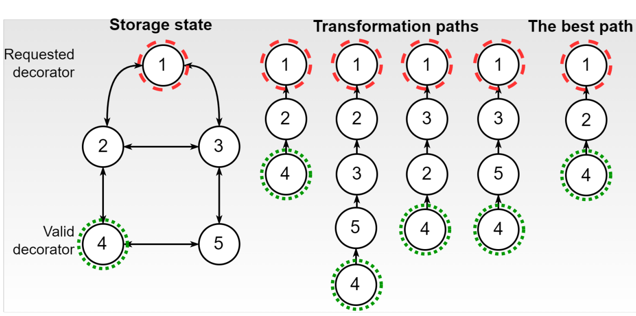

Figure 4: Vector storage transformation process. Green is valid format. Red is requested format. No highlight is currently invalid format.

3.5 Differences with GraphBLAS standard

To be clear, the proposed solution is not an implementation of GraphBLAS C or C++ API. The design of the library uses only the concepts described in the standard. However, in the proposed solution, the signatures and semantics of some of the operations have been changed. The API has been made more verbose and explicit. In particular, the handling of *zero* elements and *masking* are made cleaner for the end user. The library interprets data simply as collections of bytes, without mathematical semantics and identity elements. Identity element must be explicitly passed by the user where required.

4 IMPLEMENTATION DETAILS

4.1 Core

4.2 Linear Algebra Operations

4.3 Graph Algorithms

5 EVALUATION

For performance analysis of the proposed solution, we evaluated a few most common graph algorithms using real-world sparse matrix data. As a baseline for comparison we chose LAGraph [7] in connection with SuiteSparse [2] as a CPU tool, Gunrock [6] and GraphBLAST [8] as a Nvidia GPU tools. Also, we tested algorithms on several devices with distinct OpenCL vendors in order to validate the portability of the proposed solution. In general, these evaluation intentions are summarized in the following 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?
- **RQ3** What is the performance of the proposed solution in the CPU vs. integrated GPU comparison mode?

5.1 Evaluation Setup

For evaluation of RQ1, we use a PC with Ubuntu 20.04 installed, which has 3.40Hz Intel Core i7-6700 4-core CPU, DDR4 64Gb RAM, Intel HD Graphics 530 integrated GPU, and Nvidia GeForce GTX 1070 dedicated GPU with 8Gb on-board VRAM. **For evaluation of RQ2**, we use a PC with Ubuntu 22.04 installed, which has 4.70Hz AMD Ryzen 9 7900x 12-core CPU, DDR4 128 GB RAM, AMD GFX1036 integrated GPU, and either Intel Apc A770 flux dedicated GPU with 8GB on-board VRAM or AMD Radeon Vega Frontier Edition dedicated GPU with 16GB on-board VRAM. **For evaluation of RQ3**, the first PC with Intel CPU and integrated GPU and the second PC with AMD CPU and integrated GPU are used.

Programs using CUDA were compiled with GCC v8.4.0 and Nvidia NVCC v10.1. Release mode and maximum optimization level were enabled for all tested programs. Data loading time, preparation, format transformations, and host-device initial communications are excluded from time measurements. All tests are averaged across 10 runs. The deviation of measurements does not exceed the threshold of 10 percent. Additional warm-up run for each test execution is excluded from measurements.

Table 1: Dataset description.

Graph	Vertices	Edges	Avg	Sd	Max
coAuthorsCit	227.3K	1.6M	7.2	10.6	1.4K
coPapersDBLP	540.5K	30.5M	56.4	66.2	3.3K
amazon2008	735.3K	7.0M	9.6	7.6	1.1K
hollywood2009	1.1M	112.8M	98.9	271.9	11.5K
comOrkut	3.1M	234.4M	76.3	154.8	33.3K
citPatents	3.8M	33.0M	8.8	10.5	793.0
socLiveJournal	4.8M	85.7M	17.7	52.0	20.3K
indochina2004	7.4M	302.0M	40.7	329.6	256.4K
belgiumosm	1.4M	3.1M	2.2	0.5	10.0
roadNetCA	2.0M	5.5M	2.8	1.0	12.0
rggn222s0	4.2M	60.7M	14.5	3.8	36.0
rggn223s0	8.4M	127.0M	15.1	3.9	40.0
roadcentral	14.1M	33.9M	2.4	0.9	8.0

5.2 Graph Algorithms

For preliminary study *breadth-first search* (BFS), *single-source shortest paths* (SSSP), *page rank* (PR) and *triangles counting* (TC) algorithms were chosen. Implementation of those algorithms is used from official examples packages of tested libraries with default parameters. Compared tools are allowed to make any optimizations as long as the result remains correct. The graph vertex with index 1 is set as the initial traversal vertex in the algorithms BFS and SSSP for all tested instruments and all tested devices.

5.3 Dataset

Thirteen matrices with graph data were selected from the Sparse Matrix Collection at University of Florida [3]. Information about graphs is summarized in Table 1. Average, sd and max metrics relate to out degree property of the vertices. All datasets are converted to undirected graphs. Self-loops and duplicated edges are removed. For SSSP weights are initialized using pseudo-random generator with uniform $[0, 1]$ distribution of floating-point values.

Graphs are roughly divided into two groups. The first group represents relatively dense graphs, where the number of edges per node is sufficient on average to effectively load the GPU with useful work. The second group represents relatively sparse graphs, where the average vertex degree is below the typical GPU vector register size, and the search depth reaches hundreds of hoops. Graphs are sorted in ascending order by the number of vertices within each group.

5.4 Results Summary

Table 2 presents results of the evaluation and compares the performance of Spla against other Nvidia GPU tools and uses as a baseline LaGraph CPU tool. Table 3 presents result of the portability analysis of the proposed solution. It depicts runtime timings of the proposed solution on discrete GPUs of distinct vendors. Cell left empty with *none* if tested tool failed to analyze graph due to *out of memory* exception.

Table 2: Performance comparison of the proposed solution. Time in milliseconds (lower is better).

Dataset	GB	GR	LG	SP
BFS				
coAuthorsCit	5.0	1.9	6.3	6.9
coPapersDBLP	19.9	4.5	18.0	11.5
amazon2008	8.3	3.3	20.4	8.1
hollywood2009	64.3	20.3	23.4	20.3
belgiumosm	200.6	84.4	138.0	181.2
roadNetCA	116.3	32.4	168.2	101.7
comOrkut	none	205.0	40.6	53.2
citPatents	30.6	41.3	115.9	35.1
rggn222s0	367.3	95.9	1228.1	415.3
socLiveJournal	63.1	61.0	75.5	57.1
indochina2004	none	33.3	224.6	328.7
rggn223s0	615.3	146.2	2790.0	754.9
roadcentral	1383.4	243.8	1951.0	710.2
SSSP				
coAuthorsCit	14.7	2.1	38.9	10.3
coPapersDBLP	118.6	5.6	92.2	25.7
amazon2008	43.4	4.0	90.0	21.7
hollywood2009	404.3	24.6	227.7	57.5
belgiumosm	650.2	81.1	1359.8	240.9
roadNetCA	509.7	32.4	1149.3	147.9
comOrkut	none	219.0	806.5	241.0
citPatents	226.9	49.8	468.5	129.3
rggn222s0	21737.8	101.9	4808.8	865.4
socLiveJournal	346.4	69.2	518.0	189.5
indochina2004	none	40.8	821.9	596.6
rggn223s0	59015.7	161.1	11149.9	1654.8
roadcentral	13724.8	267.0	25703.4	1094.3
PR				
coAuthorsCit	1.6	10.0	24.3	3.2
coPapersDBLP	17.6	120.2	297.6	6.1
amazon2008	5.2	40.6	89.8	5.5
hollywood2009	62.9	559.5	1111.2	32.4
belgiumosm	4.4	22.9	167.6	9.4
roadNetCA	6.6	37.7	225.8	19.6
comOrkut	none	2333.6	5239.0	103.3
citPatents	27.0	686.1	1487.0	38.3
rggn222s0	45.2	320.0	563.5	26.6
socLiveJournal	none	445.9	2122.5	112.0
rggn223s0	none	662.7	1155.6	103.4
roadcentral	none	408.8	2899.9	172.0
TC				
coAuthorsCit	2.3	2.0	17.3	3.0
coPapersDBLP	105.2	5.3	520.8	128.4
amazon2008	11.2	3.9	73.9	10.8
roadNetCA	6.5	32.4	46.0	7.7
comOrkut	1776.9	218.0	23103.8	2522.0
citPatents	65.5	49.7	675.0	54.5
socLiveJournal	504.3	69.2	3886.7	437.8
rggn222s0	73.2	101.3	484.5	77.7
rggn223s0	151.4	158.9	1040.1	204.2
roadcentral	42.6	259.3	425.3	52.7

GraphBLAST (GB), Gunrock (GR), LaGraph (LG), Spla (SP).

Table 3: Portability of the proposed solution. Time in milliseconds (lower is better).

Dataset	Intel Arc	AMD Vega	Nvidia Gtx
BFS			
coAuthorsCit	12.8	8.3	6.9
coPapersDBLP	10.8	14.9	11.5
amazon2008	12.3	12.6	8.1
hollywood2009	15.3	26.7	20.3
belgiumosm	627.5	292.4	181.2
roadNetCA	265.5	259.8	101.7
comOrkut	33.2	63.6	53.2
citPatents	21.0	30.3	35.1
rggn222s0	825.3	1259.7	415.3
socLiveJournal	43.0	85.8	57.1
indochina2004	220.6	573.4	328.7
rggn223s0	1245.5	2519.6	754.9
roadcentral	1864.9	1680.8	710.2
SSSP			
coAuthorsCit	18.3	10.4	10.3
coPapersDBLP	22.9	27.7	25.7
amazon2008	23.4	22.2	21.7
hollywood2009	44.6	56.2	57.5
belgiumosm	1085.9	454.8	240.9
roadNetCA	447.3	422.5	147.9
comOrkut	79.7	111.5	241.0
citPatents	49.8	78.4	129.3
rggn222s0	1378.8	924.3	865.4
socLiveJournal	82.7	120.7	189.5
indochina2004	366.2	519.0	596.6
rggn223s0	1880.2	1201.4	1654.8
roadcentral	3176.3	2848.8	1094.3
PR			
coAuthorsCit	3.9	1.0	3.2
coPapersDBLP	5.7	6.1	6.1
amazon2008	25.2	4.0	5.5
hollywood2009	22.6	32.4	32.4
belgiumosm	10.2	7.1	9.4
roadNetCA	10.8	15.7	19.6
comOrkut	31.9	46.6	103.3
citPatents	12.3	21.3	38.3
rggn222s0	13.4	22.4	26.6
socLiveJournal	210.0	64.2	112.0
rggn223s0	38.6	57.2	103.4
roadcentral	57.9	89.6	172.0
TC			
coAuthorsCit	4.6	2.2	3.0
coPapersDBLP	57.6	106.2	128.4
amazon2008	6.9	8.5	10.8
roadNetCA	5.4	5.4	7.7
comOrkut	1533.5	3267.6	2522.0
citPatents	25.9	39.8	54.5
socLiveJournal	280.6	420.3	437.8
rggn222s0	21.0	57.8	77.7
rggn223s0	56.7	123.2	204.2
roadcentral	14.5	34.6	52.7

Distinct devices. Performance in not for comparison.

Table 4: Integrated GPU mode performance comparison of the proposed solution. Time in milliseconds (lower is better).

Dataset	Intel		AMD	
	LG	SP	LG	SP
BFS				
coAuthorsCit	7.5	26.3	3.9	18.2
coPapersDBLP	18.7	57.3	12.0	54.9
amazon2008	24.6	65.0	13.5	40.0
hollywood2009	23.8	100.1	14.8	86.6
belgiumosm	131.4	536.0	60.0	527.6
roadNetCA	173.2	461.8	100.8	339.7
comOrkut	41.6	341.4	25.2	269.4
citPatents	126.9	371.6	61.3	217.7
rggn222s0	1288.0	1959.9	644.6	1821.7
socLiveJournal	75.0	429.8	41.6	301.6
indochina2004	228.5	1424.8	137.0	1445.1
rggn223s0	2850.8	3647.2	1403.9	3701.3
roadcentral	2087.8	3196.3	767.2	2670.3
SSSP				
coAuthorsCit	40.5	42.5	29.2	40.5
coPapersDBLP	92.9	141.8	48.9	181.6
amazon2008	97.4	114.4	48.3	131.3
hollywood2009	236.7	337.9	93.8	507.4
belgiumosm	1383.2	854.3	588.9	845.7
roadNetCA	1174.2	721.7	712.7	482.9
comOrkut	822.9	1420.5	214.8	1699.5
citPatents	488.3	669.4	171.4	897.3
rggn222s0	4919.1	5928.3	2845.6	4952.9
socLiveJournal	534.7	1007.7	185.3	1205.1
indochina2004	837.1	3708.3	345.5	3971.8
rggn223s0	11375.6	11567.8	6099.6	9899.7
roadcentral	26314.1	4887.0	7867.2	3102.0
PR				
coAuthorsCit	25.3	5.0	17.6	5.9
coPapersDBLP	302.3	26.2	154.5	39.0
amazon2008	93.0	17.5	36.0	22.4
hollywood2009	1109.8	179.9	531.7	300.7
belgiumosm	178.9	35.0	45.1	29.4
roadNetCA	236.9	86.9	67.6	86.2
comOrkut	4458.5	531.9	959.6	701.4
citPatents	1559.9	159.8	277.4	195.7
rggn222s0	576.7	145.9	275.1	270.2
socLiveJournal	2181.0	449.7	520.5	630.9
rggn223s0	1187.0	309.3	617.2	605.3
roadcentral	2995.8	461.4	993.7	409.8
TC				
coAuthorsCit	17.3	8.3	5.2	28.3
coPapersDBLP	534.1	604.2	129.4	1682.3
amazon2008	75.4	34.5	22.2	126.6
belgiumosm	28.1	23.4	11.3	67.8
roadNetCA	47.7	35.2	21.5	105.6
citPatents	693.1	247.6	170.5	589.3
rggn222s0	495.2	481.3	177.7	1218.1
roadcentral	438.8	355.8	176.6	679.7

LaGraph (LG), Spla (SP).

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?

RQ3. What is the performance of the proposed solution in the CPU vs. integrated GPU comparison mode?

6 CONCLUSION

We presented a generalized sparse linear algebra framework with vendor-agnostic GPUs accelerated computations.

REFERENCES

[1] Miguel E. Coimbra, Alexandre P. Francisco, and Luís Veiga. 2021. An analysis of the graph processing landscape. *Journal of Big Data* 8, 1 (April 2021). https://doi.org/10.1186/s40537-021-00443-9

[2] 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 (Dec. 2019), 25 pages. https://doi.org/10.1145/3322125

[3] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (dec 2011), 25 pages. https://doi.org/10.1145/2049662.2049663

[4] Jeremy Kepner, Peter Aaltonen, David Bader, Aydin Buluç, Franz Franchetti, John Gilbert, Dylan Hutchison, Manoj Kumar, Andrew Lumsdaine, Henning Meyerhenke, Scott McMillan, Carl Yang, John D. Owens, Marcin Zalewski, Timothy Mattson, and Jose Moreira. 2016. Mathematical foundations of the GraphBLAS. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–9. https://doi.org/10.1109/HPEC.2016.7761646

[5] Farzad Khorasani, Keval Vora, Rajiv Gupta, and Laxmi N. Bhuyan. 2014. CuSha: Vertex-Centric Graph Processing on GPUs. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing* (Vancouver, BC, Canada) *(HPDC '14)*. Association for Computing Machinery, New York, NY, USA, 239–252. https://doi.org/10.1145/2600212.2600227

[6] Yuechao Pan, Yangzihao Wang, Yuduo Wu, Carl Yang, and John D. Owens. 2017. Multi-GPU Graph Analytics. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 479–490. https://doi.org/10.1109/IPDPS.2017.117

[7] Gábor Szárnyas, David A. Bader, Timothy A. Davis, James Kitchen, Timothy G. Mattson, Scott McMillan, and Erik Welch. 2021. LAGraph: Linear Algebra, Network Analysis Libraries, and the Study of Graph Algorithms. arXiv:2104.01661 [cs.MS]

[8] Carl Yang, Aydin Buluc, and John D. Owens. 2019. GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU. arXiv:1908.01407 [cs.DC]

[9] Peter Zhang, Marcin Zalewski, Andrew Lumsdaine, Samantha Misurda, and Scott McMillan. 2016. GBTL-CUDA: Graph Algorithms and Primitives for GPUs. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 912–920. https://doi.org/10.1109/IPDPSW.2016.185

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009