Evaluating multi-core graph algorithm frameworks

Zawadi Svela
Department of Computer Science
Norwegian University of Science and Technology
z.b.svela@gmail.com

Abstract

Multi-core and GPU-based systems offer unprecedented computational power. They are, however, challenging to utilize effectively, especially when processing irregular data such as graphs. Graphs are of great interest, as they are now used to model geographic-, social- and neural networks. Several interesting programming frameworks for graph processing have therefore been developed these past few years.

In this work, we highlight the strengths and weaknesses of the Galois, GraphBLAST, Gunrock and Ligra graph frameworks through benchmarking their single source shortest path (SSSP) implementations using the SuiteSparse Matrix Collection. Tests were done on an Nvidia DGX2 system, except for Ligra, which only provides a multi-core framework. D-IrGL, built on Galois, also provided a multi-GPU option for SSSP. We also look at program size, documentation and overall ease of use.

High performance generally comes at the price of high complexity. D-IrGL shows its strength on the very largest graphs, where it achieved the best run-time, while Gunrock processed most other large sets the fastest. However, GraphBLAST, with a relatively low-complexity interface, achieves the greatest median throughput across all our test cases. This despite that its SSSP implementation size is only 1/10th of Gunrock, which for our tests has the highest peak throughput and the fastest run-time in most cases. Ligra had less computational resources available, and consequently performed worse in most cases, but it is also a very compact and easy to use framework. Futher analyses and some suggestions for future work are also included.

1 Introduction

Graphs are an ever useful modelling tool for modelling data in countless of situations, including geographic-, social-, and neural networks. The demand for processing is increasing, but Moore's Law is not keeping up. Thus, highly parallel programs are needed for large-scale graph processing. GPUs (Graphical Processing Units) are also growing in popularity, as they have a much higher throughput potential than CPU-only systems.

Exposing parallelism in irregular algorithms, such as graph algorithms, can be difficult, and this challenge is increased when working with GPUs, which favors highly regular computations. Specialized libraries and frameworks can help in this regard.

In this work, we study four different graph algorithm frameworks, using their SSSP implementations as a reference point. Galois [1], Gunrock [2] and Ligra [3] were

This paper was presented at the NIK-2020 conference; see http://www.nik.no/.

chosen because they are well-known frameworks in the area of graph algorithms, and the GraphBLAS [4] implementation GraphBLAST [5] was added because of its novel programming model. In addition, they also represent a variety of complexity levels. From the larger Galois project, we looked at the D-IrGL [6] system specifically, which targets heterogeneous and distributed systems. Here, "Galois" will refer to the framework and the programming model, while "D-IrGL" will refer to the code that was run and tested.

SSSP was chosen for our benchmarks since it is both an important graph primitive which is simple to outline and because it has room for many implementation differences. The specific algorithm can thus play to the strengths of frameworks, regardless of the programming model they use.

The goal of this work is to give a comparison of four different frameworks, in order to help evaluate the trade-offs between both system complexity (CPU/GPU/multi-GPU) and the frameworks that facilitate programming for these devices.

Related works include a taxonomy and categorization of different parallel graph algorithm frameworks by Heidari et al. [7]. Petterson's [8] survey is similar to our own, but with a a different primitive and selection of frameworks. Both of these works included Gunrock and Galois in their comparisons, but not D-IrGL, Ligra or GraphBLAST.

2 Background

The following sections introduce the four main frameworks benchmarked in this study: Galois [1], Gunrock [2], Ligra [3] and GraphBLAS [4]. We also include some of the abstractions used by the frameworks we evaluated, including the operator formulation used by Galois, the breath-first-based Frontier-based view associated with Gunrock and Ligra, which is also implicitly used in GraphBLAS, as well as some of the basic linear algebra terminology associated with graph representations.

Galois

Galois is a framework built around Tao-analysis, [1], and uses the *operator formulation* of algorithms to reveal *amorphous data-parallelism*, see Section 3. As such, the key features Galois provides are concurrent data structures and parallel loops that allow generation of new work items/loop iterations as the loop is running. They also provide support for priority and ordering between work items.

D-IrGL is one of the more recent additions to the continually growing family of systems associated with Galois. It is the combination of the communication substrate Gluon [6] with the GPU-targeted intermediate representation IrGL [9], and it facilitates distributed heterogeneous graph applications. These ideas were developed further into the Abelian compiler [10], which produces Gluon+IrGL code from ordinary Galois code written for CPU. The complete system effectively turns Galois into a portable framework for multi-core, distributed and potentially heterogeneous graph algorithms.

SSSP implementation. D-IrGL uses the *Chaotic Relaxation* algorithm for SSSP [11]. The source vertex begins as active, relaxes the labels of its neighbours and activates them. Now, in any order, the same procedure is performed on active vertices, relaxing labels and activating new vertices as necessary, until the algorithm converges. Because D-IrGL computes in a bulk-synchronous parallel manner, the pattern of computations would resemble a parallel Bellman-Ford implementation like in Ligra or GraphBLAST.

Gunrock

Gunrock is a frontier-based framework (see Section 3) for single-node GPU accelerated systems. In its original implementation, it provided three functions for processing frontiers: *Advance* for traversal, *compute* for label updates and *filter*. A few more specialized operators where added later.

SSSP Implementation. Gunrock uses two operators for its SSSP implementation, *advance* and *filter*. When advancing, they use an atomic minimum function to update the target vertex distance, and *filter* removes any redundant vertices from the frontier. They also utilize a two-level priority queue to process more relevant vertices will be processed first when the frontier size grows large. An older version of Gunrock has multipe-GPUs support for SSSP, but we decided to focus on its recent version which has a new programming model, but only has single-GPU support for SSSP.

Ligra

Ligra is a light-weight frontier-based framework for CPUs. It provides only two functions, *edgeMap* and *vertexMap* and a *vertexSubset* type for the frontier. EdgeMap is the advancing operator, and applies a given function to all edges out of the frontier given a condition is met. VertexMap is similar to Gunrock's *filter* function. Both functions have precise mathematical definitions provided in the paper, and as a consequence Ligra is a very compact framework, which is explored further in Section 6.

SSSP implementation. Ligra implements a straight-forward Bellman-Ford algorithm. As with any frontier-based framework, it will only relax edges incident to the vertex frontier, in contrast all edges as in the original Bellman-Ford algorithm.

GraphBLAS

GraphBLAS [4] is an open standard for sparse linear algebra operations targeting graph algorithms. Inspired by the BLAS standard, it seeks to be portable and specifies the semiring structure, see Section 3, as well as different linear algebra operations such as matrix-vector multiplication and element-wise addition. There is a reference implementation in SuiteSparse, as well as mutliple other implementations.

GraphBLAST

GraphBLAST [5] is a implementation of GraphBLAS targeting Nvidia-GPUs. It uses an extension of a technique employed in a BFS-implementation by Beamer *et al.* [12], who observed that when the frontier of a breadth-first search (BFS) is large enough, it is beneficial to pull information from the vertices outside the frontier rather than the frontier vertices pushing the search forward. The dichotomy between pull and push-style computation matches dense and sparse vector-matrix multiplication, and not just for BFS. This is exploited in GraphBLAST, and the authors call this technique *generalized direction-optimization*.

SSSP Implementaion. GraphBLAST's SSSP algorithm is similar to that of Ligra, as linear algebra exhibits a very similar compuitational pattern. In addition to direction-optimization, they employ *sparsification* of the frontier. An element-wise minimum-operation is performed between the the frontier and the distance label vector to rid the frontier of any vertices that did not see any improvement in their distance.

3 Abstractions

This section will present the different abstractions utilized by the frameworks. There are many different abstractions for exposing parallelism in graph programs. They are given high importance here as they are how the user conceptualizes and models their algorithm. They are also interesting to look at separately because they might be implemented differently by different frameworks.

Operator formulation

Used by Galois and key in Tao-analysis [1], the operator formulation views an algorithm as actions on a data structure. The formulation is general enough to apply to any structure, but the primary goal is to reveal parallelism in irregular computations such as graph algorithms, so that will be the focus here.

An element in a graph ready for computation is called *active*. The elements read or written to by an activity is called the *neighbourhood*, and is key to performing activities in parallel, as they are the potential source of race conditions. Algorithms were activities can activate new elements are called *data-driven*.

A benefit to this model is how it decouples the generation of activities from any ordering or dependencies they might have. It exposes what Pingali *et al.* refers to as *amorphous data-parallelism*, and if one ensures that no computation is committed until a safe state is reached, possibly by issuing roll-backs, activities can be performed in parallel and in any order. This leaves room for many optimizations "behind the scenes" that the user of this kind of system does not need to interact with directly.

Frontier-based

One of the most popular abstraction is the frontier, which is based on the pattern of a breadth-first search. The frontier represents the active elements in the graph, usually vertices, and algorithms are constructed by iteratively applying operators on the frontier until it empties/converges. The most important operator is one that advances the frontier, producing a new frontier with neighbours from the previous one. In addition, one needs the possibility of updating internal values of vertices and filter out frontier elements. For many algorithms, this would yield a procedure like in Figure 1, the pattern of a breadth-first-like search. Among those frameworks studied in this work, this abstraction is used by Gunrock, Ligra and implicitly in GraphBLAST.

```
Initialize frontier
while frontier not empty do
    Advance frontier to neighbours
    Update label on newly discovered vertices
    Filter out previously discovered vertices
end while
```

Figure 1: Outline of frontier-based algorithms.

Parallelism in this abstraction is available in all operators. As noted by Wang *et al.* [2], the advance operator presents the most challenges, as it is irregular, some vertices can have vastly different degrees than others.

Linear Algebra

If a graph is represented as a matrix, many graph traversal operations similar to the ones mentioned in Section 3 can be described in the language of linear algebra. Given an adjacency matrix A and a vector representing a frontier x, the matrix-vector product $A^T x$ produces the same pattern of operations as advancing through the graph in a frontier-based graph framework. One does, however, need to adjust the actual operators used, which is expressed through semirings.

A semiring is an algebraic structure consisting of an additive operator, a multiplicative operator and a domain, as well as "zero" and identity ("one") values. Examples of different semi-rings can be found in Table 1, along with what kind of primitive a matrix-vector multiplication would imply. This is perhaps the most important feature of this abstraction. By using semirings, it allows for re-use of patterns and optimizations from linear algebra when constructing graph algorithms.

Table 1: A few example semirings.

Result	operators $\bigoplus$	operators $\bigotimes$	domain	0	1
Standard arithmetic	$+$	$\times$	R	0	1
Breadth-first search	or	and	$\{0,1\}$	0	1
Single source shortest path	*min*	$+$	$\{-\infty, R\}$	$-\infty$	0

4 Benchmarking the Frameworks

In order to evaluate and compare the performance of the different frameworks, we ran their SSSP implementations with a suite of different data sets ranging from 64 to 4,294,966,740 edges. These sets span a variety of different domains with different graph characteristics. This section describes our test set-up, the methodology behind the experiments as well as the results we obtained. Arbitrarily, the results are ordered Gunrock, D-IrGL, GraphBLAST, then Ligra, an artifact of how the tests were run.

Testbed Systems

All GPU based frameworks were tested on a Nvidia DGX-2 node. This node contains two boards of 8 V100 GPUs each. Each board also contains 6 NVSwitch interconnects.

Ligra, the only CPU-based framework tested, was tested on a Supermicro SuperServer 6049GP-TRT with 2 CPUs, Intel Xeon Gold 6230 SP - 20-Core. All cores were capable of hyperthreading, and all of the 80 available threads were utilized for the experiments.

Price comparison. As a point of reference, the CPU resources used by Ligra has a standalone price of ~4.000USD at the time of writing and a single V100 GPU, which most configurations of the other programs used, has a price of ~8.000USD.

Bechmarking Methodology

One SSSP implementation was compiled from each framework and used to process 128 different graphs. Run-time reported from each program excluded data-movement. Each program was run at least 3 times for each graph to overcome variance and the best time was considered for comparisons and evaluation.

A single configuration was used for each program, except for D-IrGL which was tested on multiple GPUs. We indicated to the programs whether the graph was undirected, but

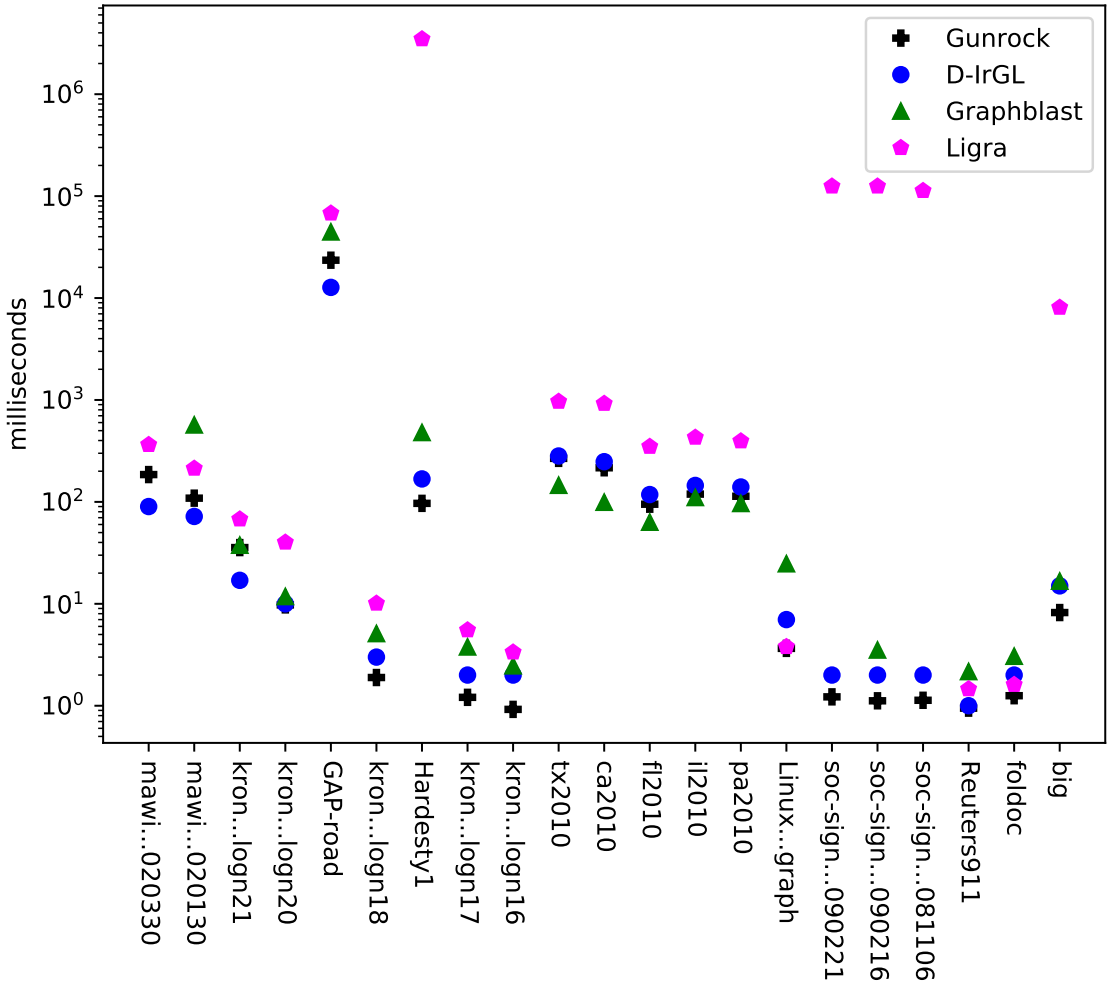


Figure 2: Runtime in milliseconds for the largest graphs with $> 10,000$ visited vertices. Only the 5 largest "xx2010" are shown.

Table 2: Number of best times per framework.

Gunrock	56	D-IrGL (Galois)	4
Graphblast	47	Ligra	5

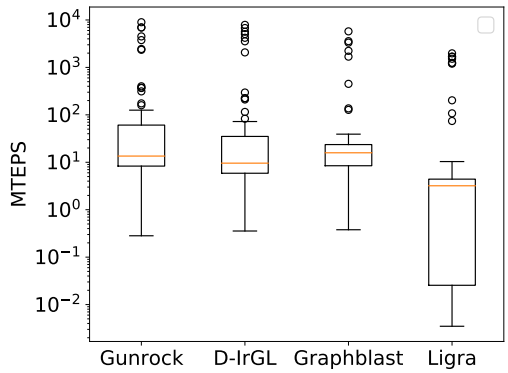
not any other graph characteristics. Gunrock reports the number of visited vertices for each graph, and this was used to filter results and estimate throughput.

Data sets

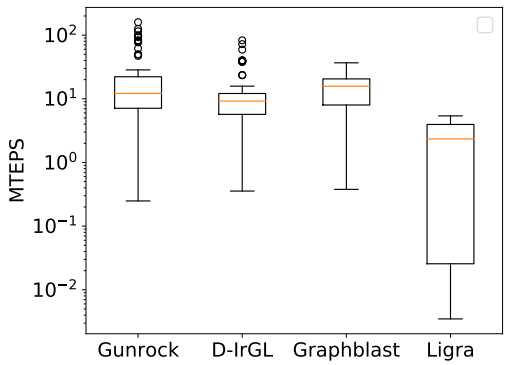
All data sets were pulled from the SuiteSparse Matrix collection [13]. Only square integer matrices with the keyword "weighted" were considered. Ligra does not support floating point values, and matrices representing unweighted graphs would not be suited for comparing SSSP implementations. This left 128 graphs.

5 Benchmarking results

This section presents and evaluates the results of the experiments. Two metrics were deemed the most relevant, run-time on large graphs, and average throughput.



(a) Visited vertices >= 30. 110 out of 128 graphs.



(b) Complete traversal only. 83 out of 128 graphs.

Figure 3: Millions of traversed edges per second estimates for different subsets of the results. Graphs where Gunrock crashed are excluded.

Table 3: Peak throughput for each program.

Program	Peak Throughput
Gunrock	8969
Galois	7885
GraphBLAST	5754
Ligra	1980

Relative run-time

Figure 2 shows a selection of the running time for the different frameworks on largest graphs with more than 10,000 visited vertices. The general trend ranking the relative performance is Gunrock, D-IrGL, GraphBLAST and then Ligra. This is not wholly unexpected. Gunrock is specifically developed specifically for high performance and for single-node systems. D-IrGL is also developed for high-performance, but for potentially distributed systems. GraphBLAST and Ligra are light-weight frameworks with a stronger focus on accessibility, and additionally Ligra only runs on CPU and therefore had less computational resources available than the GPU-based programs.

However, as can be seen in Table 2, the aforementioned trend is not universal. D-IrGL outperforms Gunrock in four of the five largest graphs. Among these there are both scale-free graphs and a road network, so results does not seem to be tied to specific properties of the graphs. Ligra also outperformed Gunrock on some smaller graphs (less than 10,000 edges), as well as the "geom" data set with approximately 20,000 edges, which for context is roughly 1/4th the size of the smallest graph present in Figure 2. Lastly, GraphBLAST outperformed the other programs in a significant number of cases, which is further discussed in Section 5.

Throughput

In addition to runtime on specific graphs, we wanted to measure the throughput efficiency of each framework to get a somewhat normalized metric to compare the programs by. We used an estimate of millions of edges traversed per second (MTEPS) as our throughput metric, visualized in the boxplots of Figure 3. The number of edges processed by each program is estimated based on the number of visited vertices reported by Gunrock, which

would be the same for any SSSP algorithm, and the average degree of the graph. This is not a perfect estimate, as different programs process different amounts of edges, and the average degree might not reflect the connected component explored, but it still gives a sense of how efficient the programs are in relation to the graph size.

Using this metric, one can see that GraphBLAST actually outperforms the rest in terms of its median MTEPS across all graphs, Figure 3a. This is even clearer when looking at only complete traversals 3b. If we only look at peak throughput, however, see Table 3, Gunrock and Galois again take the lead, with peaks that are respectively 56 and 37 percent better than that of GraphBLAST. Ligra has a very high variance in its throughput. It is unclear what causes this variance, and it might either be hardware/CPU specific or caused by Ligra'a execution model.

Graph characteristics

In general, the graphs represented in the data sets seem to span a few staple categories of graph analysis, which allows us to test the generality of the programs. There are many scale-free graphs, both synthetic and real, and there are road-networks and mesh-like graphs. Scale-free graphs or power-law graph are frequently used as test cases and cited as challenges, [2,3,5,8], which makes sense as they have highly skewed degree distribution.

Of the 110 graphs with valid results, 50 are "xx2010" graphs from the DIMACS10 collection. These are based on US census data, and vertices and edges model respectively census blocks and their borders. These graphs have a high diameter and an even distribution of degrees, in contrast to scale-free networks that generally have a highly skewed distribution and low diameter. Interestingly, removing the "xx2010" graphs increased the median throughput of both Gunrock and D-IrGL, but the opposite was the case for GraphBLAST and Ligra. The reason for this is unknown. GraphBLAST achieved the best run-time on 47 of these graphs, but not on any other sets, which might indicate these graphs have attributes especially suited for this particular framework.

D-IrGL and multi-GPU

D-IrGL is developed for potentially distributed systems with multiple GPUs, and was tested on both 1, 2, 4 and 8 GPUs. The performance worsened with multiple GPUs compared to one, but it did manage to load the GAP-kron graph with 4 and 8 GPUs, a graph that caused all other programs and configurations to run out of memory. Because of this, large graphs is a use-case for these configurations, even if the throughput is lowered.

6 Evaluation

A framework is a toolkit for constructing new programs. As such, its quality is not just dependent on its efficiency, but also its usability. This section contains a comparison of code sizes as well as a subjective assessment of the ease of use of the different frameworks. The focus is on the available documentation, including papers published in relation to the frameworks. A summary of this section can be found in Table, 4.

Framework code sizes

The total lines of code for the different SSSP implementations are provided in Table 5. This is an important metric, as large code bases are both harder to understand, and also harder to maintain. In all cases the programs have additional features such as support for multiple runs and timing, but they still give an indication of how easy it is to understand

Table 4: Accessibility summary

	Code size	Documenation	System
D-IrGL	Very large	N/A	Multi-GPU
Gunrock	Very large	Fair	Single node, multi-GPU
GraphBLAST	Small	Good	Single GPU
Ligra	Very small	Good	Multicore CPU

Table 5: Lines of code per SSSP implementation, including per file, by code size.

Ligra	87	BellmanFord.C
GraphBLAST	138	gsssp.cu
Gunrock	293	sssp_app.cu
	361	sssp_enactor.cuh
	408	sssp_problem.cuh
Total	1062	
D-IrGL	944	sssp_push_cuda.cu
	216	sssp_push_cuda.cuh
	147	sssp_push_cuda.h
Total	1307	
Galois	494	SSSP.cpp

the given program. From these counts, it is clear that Ligra and GraphBLAST are by far the most compact frameworks, which should make them easier to understand than the others. Gunrock lies as the other extreme with large file sizes and also multiple files. D-IrGL code is constructed by the Abelian compiler and as such isn't necessarily meant to be read or interacted with directly, but the Galois CPU-implementation which D-IrGL is based on, still has a fair number of lines in its code.

Documentation

Documentation is important especially frameworks. The perspective of this assessment is after a thorough reading of relevant papers describing the frameworks, any tutorials available, and a scan of available manuals. The developers of all 4 frameworks were a great help clearing up any confusion throughout the process of this evaluation, which goes a long way for compensating for ways in which documentation might be lacking.

Galois

The online documentation of Galois contains both a comprehensive manual and a step-by-step tutorial on how to create a simple program. As Galois is a pretty complex framework, this helps in easing into the necessary concepts.

It does not incorporate all features of Tao-analysis, most notably all operators are assumed cautious. This means that ethey can not write to a data structure before executing all necessary reads. This allows Galois to use locks to detect conflicts before an operation is committed, which again means it does not need to execute any rollbacks.

D-IrGL The documentation of how to run the D-IrGL programs is good, and there were no major issues encountered. However, there is no documentation available on how to write D-IrGL programs.

GraphBLAST

GraphBLAST has minimal documentation in comparison to Gunrock and Galois. There is a single text-file in their Git repository, which contains an example SSSP implementation with comments, and a short explanation of their linear algebra operations, as well as semirings and how they relate to graph algorithms.

The GraphBLAST documentation assumes the user is comfortable with linear algebra operations and how it relates to graph algorithms. Reading the GraphBLAST paper is thus recommended to understand all the sample algorithms and make one's own programs.

Gunrock

In general, the presentation of Gunrock is quite excellent. They have an easily navigable website that provides an overview of the programming model, performance analysis, and links to publications and their GitHub code repository with notes on the state of the current build. They also provide a quick start guide for building and running the algorithm implementation they provide. Also beneficial is a thorough description of exactly what is measured and reported for the different programs, which aids benchmarking.

At the time of writing, Gunrock has recently transitioned from release 0.5 to release 1.0, which saw a great simplifications of the interface. It does however, also mean that the online documentation isn't up to date yet. Notably, there is no tutorial available on how to write new primitives with Gunrock. As Gunrock is a quite complex system, it can be quite difficult to make wholly new programs from scratch at this point. A guide for porting 0.5 application is provided, which can help, and they also provide steps to generate the newest documentation locally. The developers are fully aware of these features currently lacking. They are all described in their online roadmap under "Documentation" [1].

Ligra

Similarly to GraphBLAST, Ligra's light interface allows it to have minimal documentation. It simply presents the input format, how to get the programs up and running, and provides a short description of the functions and data structures. It is very beneficial to have read the paper presenting Ligra first, [3], because the mathematical function definitions in the paper match up quite well with the actual code. There is no tutorial-specific code provided, but the code for BFS and SSSP (Bellman-Ford) are both quite short.

7 Conclusions & Future Work

Graphs are used to model a wide variety of different data. With the increasing computational demand and system complexity, recent graph frameworks provide attractive tools for implementing graph-based methods that can be very beneficial for productivity. In this paper we presented four of the most well-known frameworks for graph algorithms, with SSSP used as the reference point to evaluate both the ease of use and their performance. All frameworks showed strengths in different use-cases. A summary of our results is followed by suggestions for future work.

If high-performance is of top priority, Gunrock had the most reliable performance, and achieved the best run-time on 56 out of 128 graphs, which included multiple different graph types. D-IrGL was generally slightly slower than Gunrock and had the best run-time on only 4 graph. However, these graphs were among the largest we tested, and D-IrGL

[1] https://github.com/gunrock/gunrock/projects/3

does in theory support arbitrarily large graphs when run on multi-device systems. D-IrGL code is not as easily re-usable, a definite weakness compared to the other programs. Gunrock also achieved the best peak throughput across all frameworks, but it is also the most complex framework out of the four. It is also currently lacking in terms of documentation of version 1.0. In addition, it's SSSP implementation indicates both high complexity and low maintainability compared to the lighter frameworks.

GraphBLAST performed surprisingly well for a relatively light-weight framework, even ignoring a large set of similar graphs where it actually outperformed all other frameworks. It achieved the best run-time on 47 graphs and had the overall best median throughput. Compared to Gunrock, it is also significantly less complex, with easier to understand documentation and an SSSP implementation of approximately 1/8th the size. It and Gunrock did however crash on a few data sets, which is a definite negative.

Ligra used ~100,000 the time of Gunrock on certain graphs, which makes it less appealing as a general case high-performance framework. Its ease of use and compactness, with SSSP code size 1/12th the size of Gunrock's, does however make appealing for pipelines with high maintainability demands or for smaller graphs.

Future work

For a more thorough comparison, one could write the same primitive in different frameworks, preferably solving a new problem rather than those already provided. Writing the exact same algorithm would not necessarily be the correct approach, as different frameworks excel at expressing different behaviours, like there were four different algorithms implemented for SSSP. Future comparisons should also include floating data sets, as this could dramatically increase the variety of the data sets.

It would have been beneficial to have a standardized suite with detailed descriptions of relevant attributes of all graphs. Two such attributes would be degree distribution and diameter, as they can greatly affect the run-time of a program.

It would also be interesting to explore exactly why GraphBLAST performed so well specifically on the "xx2010" graphs. As discussed in Section 5, this is probably related to some common attributes across these graphs, and finding the exact ones might reveal new optimization strategies for both GraphBLAST and other graph algorithms.

Acknowledgements: The author wishes to thank their advisor Prof. Anne C. Elster for helping with this write-up, and NTNU and the IDI HPC-Lab for computer resources.

References

[1] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Mendez-Lojo, D. Prountzos, and X. Sui, "The Tao of Parallelism in Algorithms," *PLDI'11*, 2011.

[2] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: a high-performance graph processing library on the GPU," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming - PPoPP '16*. Barcelona, Spain: ACM Press, 2016, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2851141.2851145

[3] J. Shun and G. E. Blelloch, "Ligra: A Lightweight Graph Processing Framework for Shared Memory," *Proceedings of the ACM SIGPLAN Symposium on Principles*

and Practice of Parallel Programming (PPoPP), pp. 135–146, 2013. [Online]. Available: https://people.csail.mit.edu/jshun/ligra.pdf

[4] A. Buluc, T. Mattson, S. McMillan, J. Moreira, and C. Yang, "Design of the GraphBLAS API for C," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Orlando / Buena Vista, FL, USA: IEEE, May 2017, pp. 643–652. [Online]. Available: http://ieeexplore.ieee.org/document/7965104/

[5] C. Yang, A. Buluc, and J. D. Owens, "GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU," *arXiv:1908.01407 [cs]*, Sep. 2019, arXiv: 1908.01407. [Online]. Available: http://arxiv.org/abs/1908.01407

[6] R. Dathathri, G. Gill, L. Hoang, H.-V. Dang, A. Brooks, N. Dryden, M. Snir, and K. Pingali, "Gluon: a communication-optimizing substrate for distributed heterogeneous graph analytics," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2018*. Philadelphia, PA, USA: ACM Press, 2018, pp. 752–768. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3192366.3192404

[7] S. Heidari, Y. Simmhan, R. N. Calheiros, and R. Buyya, "Scalable Graph Processing Frameworks: A Taxonomy and Open Challenges," *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–53, Jul. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3199523

[8] H. Pettersson, "A Survey of Parallel Breadth-First Search Frameworks for CPUs and GPUs," *Master Thesis at NTNU*, p. 41, 2018.

[9] S. Pai and K. Pingali, "A compiler for throughput optimization of graph algorithms on GPUs," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA 2016*. Amsterdam, Netherlands: ACM Press, 2016, pp. 1–19. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2983990.2984015

[10] G. Gill, R. Dathathri, L. Hoang, A. Lenharth, and K. Pingali, "Abelian: A Compiler for Graph Analytics on Distributed, Heterogeneous Platforms," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, vol. 11014, pp. 249–264. [Online]. Available: http://link.springer.com/10.1007/978-3-319-96983-1_18

[11] D. Chazan and W. Miranker, "Chaotic relaxation," *Linear Algebra and its Applications*, vol. 2, no. 2, pp. 199–222, Apr. 1969. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0024379569900287

[12] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing Breadth-First Search," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2012, pp. 1–10, iSSN: 2167-4337.

[13] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, 2011, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/2049662.2049663