A Comparison of Parallel Graph Processing Benchmarks

Samuel D. Pollard and Boyana Norris

University of Oregon
Eugene OR 97403, USA
spollard@cs.uoregon.edu
norris@cs.uoregon.edu

Abstract. The increasing popularity of large network analysis problems has led to the emergence of many parallel and distributed graph processing systems—one survey in 2014 identified over 80. Since then, the landscape has evolved; some packages have become inactive while more are being developed. Determining the best approach for a given problem is infeasible for most developers. To enable easy, rigorous, and repeatable comparison of the capabilities of such systems, we present an approach and associated software for analyzing the performance and scalability of parallel, open-source graph libraries. We demonstrate our approach on five graph frameworks: GraphMat, the Graph500, the Graph Algorithm Platform, GraphBIG, and PowerGraph using synthetic and real-world datasets. We examine previously overlooked aspects of parallel graph processing performance such as phases of execution and energy usage for three algorithms: breadth first search, single source shortest paths, and PageRank and compare our results to Graphalytics.

1 Introduction

Another issue among parallel graph processing systems is the lack of comprehensive comparisons. One possible reason is the considerable effort involved in getting each system to run: satisfying dependencies and ensuring data are correctly formatted are generally nontrivial tasks. Beyond this, there are optimizations for each system which are not exploited in a naïve implementation. Addressing these issues helps provide an environment in which no graph processing system has an advantage.

Our contributions can be summarized as follows.

–
–
–

2 Related Work

Our research is motivated by the current state of parallel graph processing. The most comprehensive survey, released in 2014, identified and taxonomized over

80 different parallel graph processing systems without including domain specific languages [5]. The systems described operate with a wide range of parallelism paradigms and target architectures such as GPU [13, 25], shared memory CPU [14, 19, 22], a combination of CPU and GPU [7], distributed filesystem based approaches [24], and distributed memory with MPI [8].

Beyond the systems described by Doekemeijer and Varbanescu, the problem has compounded with the addition of even more proprietary and open source projects such as [4, 20], distributed memory approaches such as [11]. domain-specific languages [10], distributed database querying, [21], as well as novel communication schemes [6]. At the outset, this plethora of choices makes the question, "which system is the best for my problem?" daunting.

In addition to libraries with associated APIs there has also been a propagation of "reference implementations" which implement the most common graph algorithms such as [1, 17]. Thus, even selecting a standard and a benchmark over which to compare various implementations is nontrivial. To quote Andrew Tanenbaum, "The nice thing about standards is that you have so many to choose from."

Graphalytics [3] attempts to automate the setup and execution of packages in order to compare performance. However, in order for Graphalytics to analyze a graph processing system—Graphalytics calls these platforms—a programmer must implement Java classes as wrappers to call the particular implementations, satisfy dependencies, and set environment variables. These all require some knowledge of the inner workings of each system in addition to familiarity with the Graphalytics API.

The complexity of Graphalytics obfuscates the true behavior of the algorithm. We use GraphMat, the single source shortest paths algorithm, the dota-league dataset, and the default settings as an example here. By default, Graphalytics generates an HTML report giving the runtimes for each dataset in seconds. Two separate runs for a single dataset, all else being equal, gave 5.6 seconds. However, perusing the log files reveals a more complete picture: of these 5.6 seconds, 0.19 was spent computing the shortest paths while the remaining time consisted of building the necessary data structures.

With a plugin to Graphalytics called Granula [18] one can explicitly specify a performance model to analyze specific execution behavior such as the amount of communication or runtime of particular kernels of execution. The requires in-depth knowledge of the source code and execution model which is not always available. Furthermore, creating such a model requires a high level of expertise with the given system and with Granula.

As with Graphalytics, the initial development effort is high but Granula paired with graphalytics allows automatic execution and compilation of performance results[1]. Likewise, our approach provides automatic execution and performance analysis without requiring a performance and execution model for each system.

[1] An example of Granula can be seen at `https://github.com/tudelft-atlarge/graphalytics-platforms-graphx/tree/master/granula-model-graphx`

To showcase our methods we analyze the performance of several graph processing benchmarks to facilitate the selection among such systems. Our analysis is done with minimal source code modification.

3 Experimental Setup

3.1 Graph Processing Systems

This report explores four shared memory parallel graph processing platforms. The first three are so-called "reference implementations" while the fourth is included because it has been cited as highly performant [23]. Our target is shared memory CPU processing. Other popular libraries such as the Parallel Boost Graph Library [8] are not considered here because the authors do not provide reference implementations. The systems are:

1. The Graph500[2] [16]
2. The Graph Algorithm Platform (GAP) Benchmark Suite [1]
3. GraphBIG [17]
4. GraphMat [23]

3.2 Algorithms

We consider three algorithms:

1. Breadth First Search (BFS)
2. Single Source Shortest Paths (SSSP)
3. PageRank

The canonical performance leaderboard for parallel graph processing is the Graph500 [16]. The advantage of the Graph500 is it provides standardized measurement specifications and dataset generation. The primary drawback with the Graph500 is it measures a single algorithm: breadth first search.

This report aims to add similar rigor to other graph algorithms by borrowing heavily from the Graph500 specification. The Graph500 Benchmark 1 ("Search") is concerned with two kernels: the creation of a graph data structure from an edge list stored in RAM and the actual BFS[3]. We run the BFS using 32 random roots.

One straightforward extension to BFS and our second algorithm is the Single-Source Shortest Paths algorithm (SSSP). We use the same graph and the same source vertices as in BFS.

The third algorithm is PageRank. These three algorithms are used because of their popularity; most libraries provide reference implementations. All the

[2] We used the most recent version from `https://github.com/graph500/graph500`, most similar to release 2.1.4.

[3] For a complete specification, see `http://graph500.org/specifications`

experiments performed use the developer-provided implementations because we assume the vendor will provide a more performant implementation than us. While this limits the scope of the experiments it mitigates the bias inherent in any developer's programming skills.

3.3 Machine Specifications

Table 1 shows the specifications of the research machine. The disparity between the CPU's advertised clock speed and the "CPU Clock" row is a result of the Turbo Boost technology which can increase the clock speed to a limit. We use the manufacturer's published maximum clock speeds which can be found at `http://ark.intel.com`.

CPU Model	Intel Xeon(R) E5-2699 v3 @ 2.30GHz
CPU Sockets	2
CPU Cores	72
CPU Clock	3600MHz
RAM Size	256GB
RAM Freq	1866MHz
GPU Model	GM204 [GeForce GTX 980]

Table 1. The operating system is GNU/Linux version 4.4.0-22.

3.4 Datasets

We use the Graph500 synthetic graph generator which creates a Kronecker graph [15] with initial parameters of $A = 0.57, B = 0.19, C = 0.19$, and $D = 1 - (A + B + C) = 0.05$.

The Graphalytics results in Table 2 were performed on the Dota-League dataset. This dataset contains 61,670 vertices and 50,870,313 edges. This dataset is sourced from the Game Trace Archive [9] and modified for Graphalytics[4].

4 Performance Analysis

We use the Performance Application Programming Interface (PAPI) [2] to gain access to Intel's Running Average Power Limit (RAPL), which provides access to a set of hardware counters measuring energy usage.

In Table 2 we show the results from running Graphalytics. For an explanation of each algorithm used, see [12].

From this we get only a broad overview of the two systems: GraphBIG and PowerGraph. In general, we see GraphBIG is more performant. However, results such as these are preliminary and do not show the complete picture.

[4] This dataset is available at `https://atlarge.ewi.tudelft.nl/graphalytics/`.

[How to make this a paper worth submitting: add in PowerGraph as a platform and see how it fares using our method compared to Graphalytics. Also run this on different datasets and compare results.]

	GraphMat	GraphBIG	PowerGraph
CDLP	N/A	212.5	1,167
PR	N/A	293.5	983
LCC	N/A	316.5	1,011
WCC	N/A	89	767.5
SSSP	11,298.5	6,378.5	34,654.5

Table 2. Performance results are in milliseconds. LCC is local clustering coefficient, PR is PageRank, CDLP is community detection using label propagation, and WCC is weakly connected components. At the time of this writing, Graphalytics only supports SSSP for GraphMat.

System	Load Graph	Construct Data Structure	Run BFS
Graph500	0.3474	0.3971	0.003380
GAP	2.351	0.1935	0.001393
GraphMat	0.1511	1.101	0.1031
GraphBIG	37.22	N/A	0.1528

Table 3. The above table shows times for 2^{20} verties and the times are in seconds. The Graph500 generates the graph instead of loading it into a file. GraphBIG builds the graph and reads in the file simultaneously. These results were averaged across 32 roots.

[Note that for figure Fig.5, mention time per iteration]
[Compare lines of code for implementation]

5 Conclusion

References

1. Beamer, S., Asanovic, K., Patterson, D.A.: The GAP benchmark suite. CoRR abs/1508.03619 (2015), http://arxiv.org/abs/1508.03619
2. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A portable programming interface for performance evaluation on modern processors. Int. J. High Perform. Comput. Appl. 14(3), 189–204 (Aug 2000), http://dx.doi.org/10.1177/109434200001400303
3. Capotă, M., Hegeman, T., Iosup, A., Prat-Pérez, A., Erling, O., Boncz, P.: Graphalytics: A big data benchmark for graph-processing platforms. In: Proceedings of

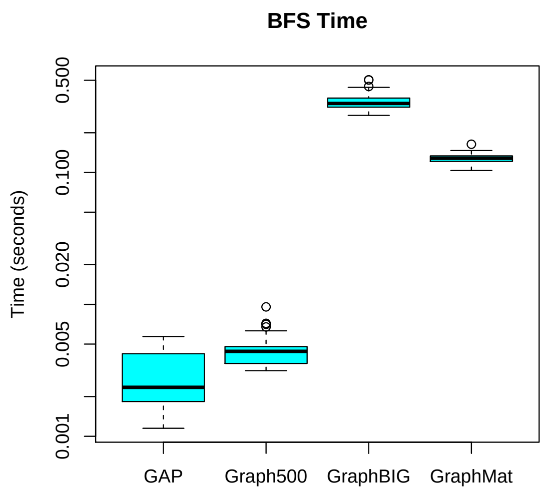

Fig. 1. The *y*-axis is logarithmic.

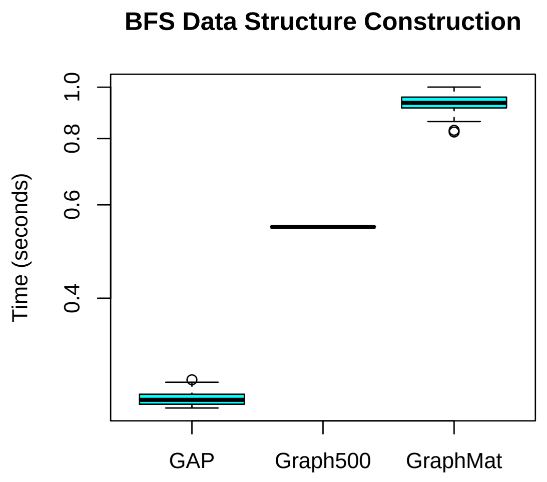

Fig. 2. The *y*-axis is logarithmic. GraphBIG reads in the file and generates the data structure simultaneously so is omitted.

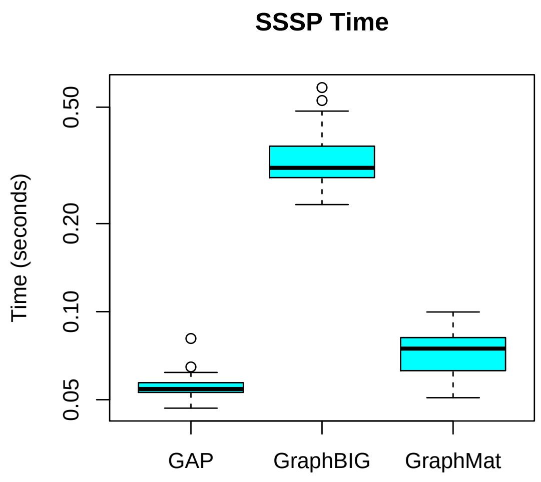

Fig. 3. The y-axis is logarithmic.

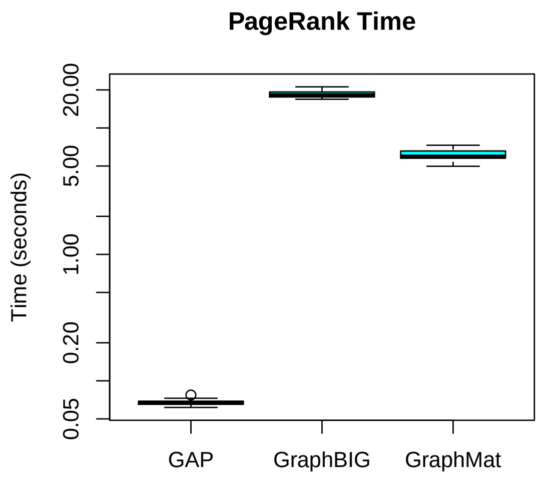

Fig. 4. The y-axis is logarithmic. GraphMat continues to run until none of the vertices' ranks change. [There is only one data point]

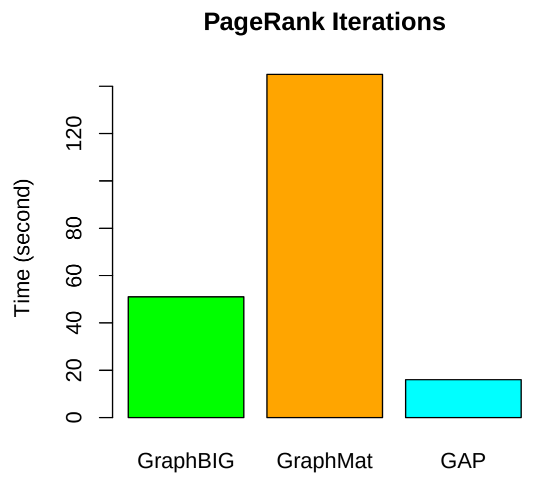

Fig. 5. The y-axis *not* logarithmic. GraphMat iterations are measured differently because of the "think like a vertex" paradigm and runs until none of the vertices' ranks change.

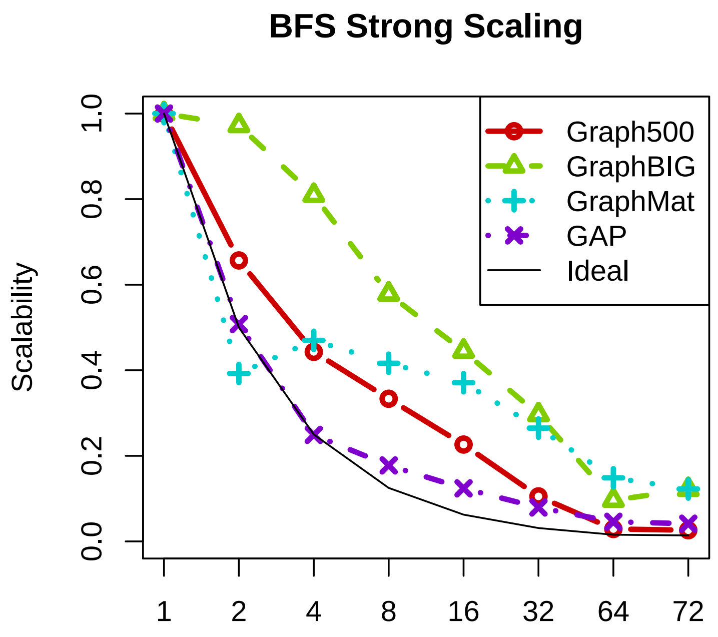

Fig. 6. Strong Scaling for 2^{20} vertices and an average of 16 edges per vertex.

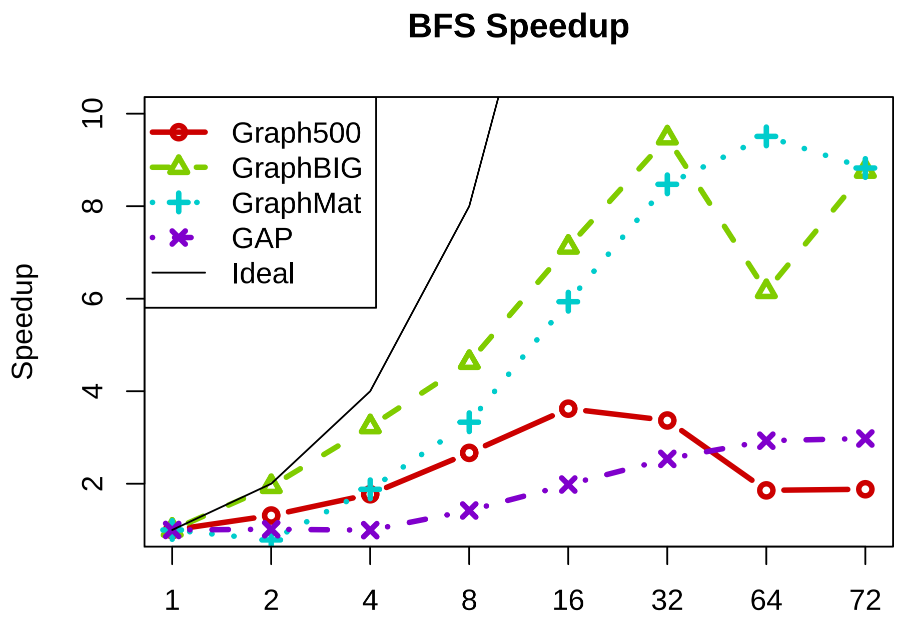

Fig. 7. Speedup.

the GRADES'15. pp. 7:1–7:6. GRADES'15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2764947.2764954
4. Cheramangalath, U., Nasre, R., Srikant, Y.N.: Falcon: A graph manipulation language for heterogeneous systems. ACM Trans. Archit. Code Optim. 12(4), 54:1–54:27 (Dec 2015), http://doi.acm.org/10.1145/2842618
5. Doekemeijer, N., Varbanescu, A.L.: A survey of parallel graph processing frameworks. Tech. rep., Delft University of Technology (2014)
6. Edmonds, N., Willcock, J., Lumsdaine, A.: Expressing graph algorithms using generalized active messages. In: Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 289–290. PPoPP '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2442516.2442549
7. Gharaibeh, A., Beltrão Costa, L., Santos-Neto, E., Ripeanu, M.: A yoke of oxen and a thousand chickens for heavy lifting graph processing. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. pp. 345–354. PACT '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2370816.2370866
8. Gregor, D., Edmonds, N., Barrett, B., Lumsdaine, A.: The parallel boost graph library. http://www.osl.iu.edu/research/pbgl (2005)
9. Guo, Y., Iosup, A.: The game trace archive. In: Proceedings of the 11th Annual Workshop on Network and Systems Support for Games. pp. 4:1–4:6. NetGames '12, IEEE Press, Piscataway, NJ, USA (2012), http://dl.acm.org/citation.cfm?id=2501560.2501566
10. Hong, S., Chafi, H., Sedlar, E., Olukotun, K.: Green-marl: A dsl for easy and efficient graph analysis. In: Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 349–362. ASPLOS XVII, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2150976.2151013
11. Hong, S., Depner, S., Manhardt, T., Van Der Lugt, J., Verstraaten, M., Chafi, H.: PGX.D: A fast distributed graph processing engine. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 58:1–58:12. SC '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2807591.2807620
12. Iosup, A., Hegeman, T., Ngai, W.L., Heldens, S., PÃľrez, A.P., Manhardt, T., Chafi, H., Capota, M., Sundaram, N., Anderson, M., Tanase, I.G., Xia, Y., Nai, L., Boncz, P.: Ldbc graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms, a technical report. Tech. rep., Delft University of Technology (2016)
13. Kang, U., Tsourakakis, C.E., Faloutsos, C.: Pegasus: A peta-scale graph mining system implementation and observations. In: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining. pp. 229–238. ICDM '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICDM.2009.14
14. Kyrola, A., Blelloch, G., Guestrin, C.: Graphchi: Large-scale graph computation on just a pc. In: Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). pp. 31–46. USENIX, Hollywood, CA (2012), https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola
15. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: An approach to modeling networks. J. Mach. Learn. Res. 11, 985–1042 (Mar 2010), http://dl.acm.org/citation.cfm?id=1756006.1756039

16. Murphy, R.C., Wheeler, K.B., Barrett, B.W., Arg, J.A.: Introducing the graph500. Tech. rep., Cray User's Group (2010)
17. Nai, L., Xia, Y., Tanase, I.G., Kim, H., Lin, C.Y.: GraphBIG: Understanding graph computing in the context of industrial solutions. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 69:1–69:12. SC '15, ACM, New York, NY, USA (2015), `http://doi.acm.org/10.1145/2807591.2807626`
18. Ngai, W.L.: Fine-grained Performance Evaluation of Large-scale Graph Processing Systems. Master's thesis, Delft University of Technology (2015)
19. Nguyen, D., Lenharth, A., Pingali, K.: A lightweight infrastructure for graph analytics. In: Proceedings of ACM Symposium on Operating Systems Principles. pp. 456–471. SOSP '13 (2013), `http://iss.ices.utexas.edu/Publications/Papers/nguyen13.pdf`
20. Perez, Y., Sosič, R., Banerjee, A., Puttagunta, R., Raison, M., Shah, P., Leskovec, J.: Ringo: Interactive graph analytics on big-memory machines. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 1105–1110. SIGMOD '15, ACM, New York, NY, USA (2015), `http://doi.acm.org/10.1145/2723372.2735369`
21. Rodriguez, M.A.: The gremlin graph traversal machine and language. CoRR abs/1508.03843 (2015), `http://arxiv.org/abs/1508.03843`
22. Shun, J., Blelloch, G.E.: Ligra: A lightweight graph processing framework for shared memory. SIGPLAN Not. 48(8), 135–146 (Feb 2013), `http://doi.acm.org/10.1145/2517327.2442530`
23. Sundaram, N., Satish, N., Patwary, M.M.A., Dulloor, S.R., Anderson, M.J., Vadlamudi, S.G., Das, D., Dubey, P.: Graphmat: High performance graph analytics made productive. Proc. VLDB Endow. 8(11), 1214–1225 (jul 2015), `http://dx.doi.org/10.14778/2809974.2809983`
24. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: A resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems. pp. 2:1–2:6. GRADES '13, ACM, New York, NY, USA (2013), `http://doi.acm.org/10.1145/2484425.2484427`
25. Zhong, J., He, B.: Medusa: Simplified graph processing on gpus. IEEE Trans. Parallel Distrib. Syst. 25(6), 1543–1552 (Jun 2014), `http://dx.doi.org/10.1109/TPDS.2013.111`