A Comparison of Parallel Graph Processing Benchmarks

Sam D. Pollard
Department of Computer and Information Science
University of Oregon
Eugene, OR, USA
Email: spollard@cs.uoregon.edu

Boyana Norris
Department of Computer and Information Science
University of Oregon
Eugene, OR, USA
Email: norris@cs.uoregon.edu

***Abstract*—In this paper we analyze the performance of four parallel graph processing systems to determine their performance and scalability using synthetic and real-world datasets. We perform analysis for three algorithms: breadth first serach, single source shortest paths, and PageRank. This paper examines previously overlooked aspects of parallel programming performance such as file I/O in addition to a more detailed performance analysis.**

I. Introduction

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 [4]. The systems described operate with a wide range of parallelism paradigms and target architectures such as GPU [24], [12], shared memory CPU [21], [13], [18], a combination of CPU and GPU [6], distributed filesystem based approaches [23], and distributed memory with MPI [7].

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 [3], [19], distributed memory approaches such as [10]. domain-specific languages [9], distributed database querying, [20], as well as novel communication schemes [5]. 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], [16]. 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."

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.

Graphalytics [2] also attempts to remedy these problems. However, in order for Graphalytics to analyze a graph processing system—Graphalytics calls these platforms—a programmer must provide Java methods to call the particular implementations and initialization shell scripts to 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.

With a plugin to Graphalytics called Granula [17] 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.

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.

II. Experimental Setup

A. 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 [22]. Our target is shared memory CPU processing. Other popular libraries such as the Parallel Boost Graph Library [7] are

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

not considered here because the authors do not provide reference implementations for all algorithms. The systems are:

1) The Graph500[2] [15]
2) The Graph Algorithm Platform (GAP) Benchmark Suite [1]
3) GraphBIG [16]
4) GraphMat [22]

B. 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 [15]. 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 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.

C. Machine Specifications

Table I 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.

D. Datasets

We use the Graph500 synthetic graph generator which creates a Kronecker graph [14] with initial parameters of $A = 0.57, B = 0.19, C = 0.19$, and $D = 1 - (A + B + C) = 0.05$.

[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

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 I
THE OPERATING SYSTEM IS GNU/LINUX VERSION 4.4.0-22.

	GraphBIG	PowerGraph
CDLP	181.67	1,226
PR	302.33	974
LCC	321.33	1,036.67
WCC	87.67	697.67
SSSP	4,061.33	29,022.3

TABLE II
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

The Graphalytics results in Table II 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[8] and modified for Graphalytics[4].

III. Performance Analysis

In Table II we show the results from running Graphalytics. For an explanation of each algorithm used, see [11].

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.

[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.]

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 III
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]

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

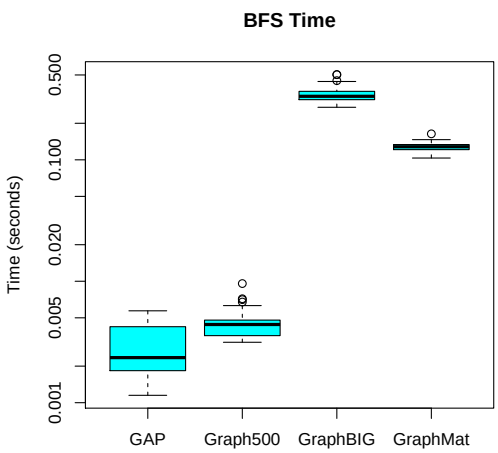

Fig. 1. The y-axis is logarithmic.

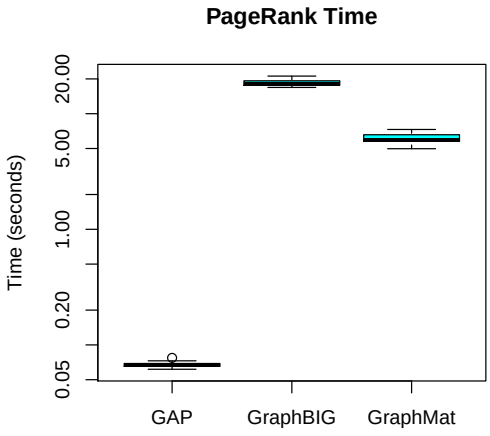

Fig. 4. The y-axis is logarithmic. GraphMat continues to run until none of the vertices' ranks change. [There is only one data point]

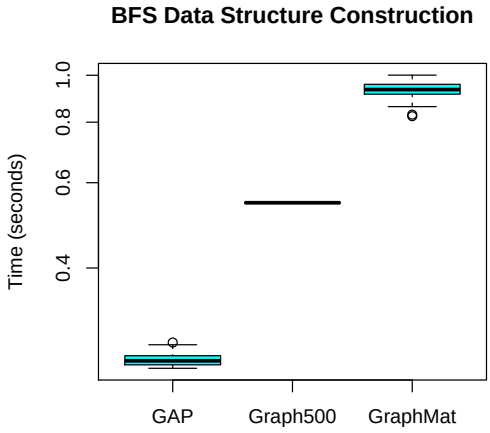

Fig. 2. The y-axis is logarithmic. GraphBIG reads in the file and generates the data structure simultaneously so is omitted.

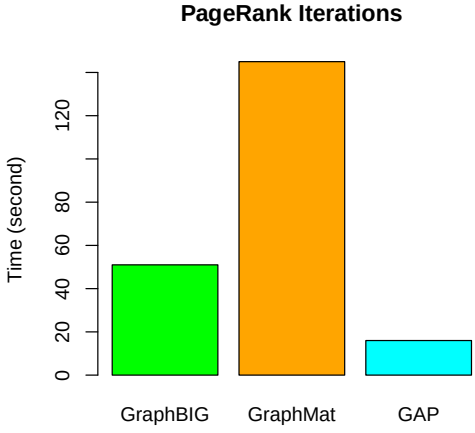

Fig. 5. The y-axis *not* logarithmic. GraphMat iterations are measured differently because of the "think like a vertex" paradigm and runs until none of the vertices' ranks change.

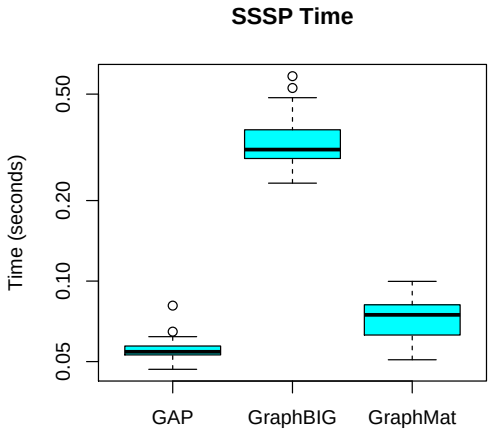

Fig. 3. The y-axis is logarithmic.

IV. Conclusion

References

[1] S. Beamer, K. Asanovic, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, 2015. [Online]. Available: http://arxiv.org/abs/1508.03619

[2] M. Capotă, T. Hegeman, A. Iosup, A. Prat-Pérez, O. Erling, and P. Boncz, "Graphalytics: A big data benchmark for graph-processing platforms," in *Proceedings of the GRADES'15*, ser. GRADES'15. New York, NY, USA: ACM, 2015, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2764947.2764954

[3] U. Cheramangalath, R. Nasre, and Y. N. Srikant, "Falcon: A graph manipulation language for heterogeneous systems," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 54:1–54:27, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2842618

[4] N. Doekemeijer and A. L. Varbanescu, "A survey of parallel graph processing frameworks," Delft University of Technology, Tech. Rep., 2014.

[5] N. Edmonds, J. Willcock, and A. Lumsdaine, "Expressing graph algorithms using generalized active messages," in *Proceedings*

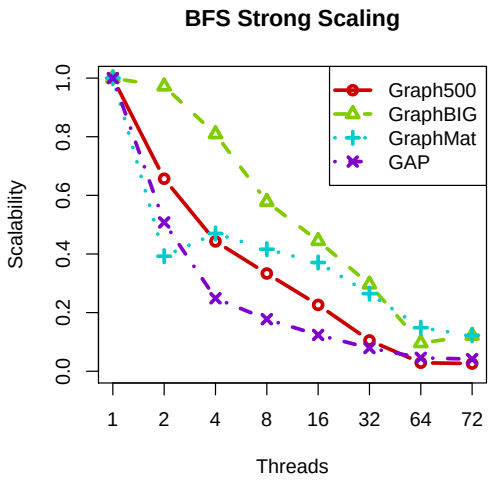

Fig. 6. Strong Scaling for 2^{20} vertices and an average of 16 edges per vertex.

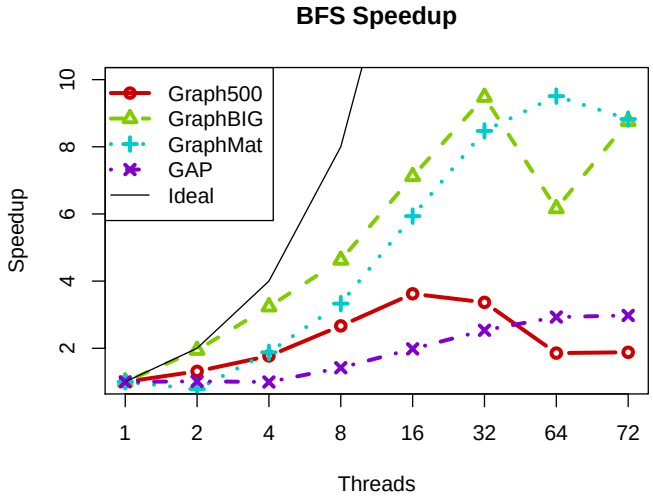

Fig. 7. Speedup.

of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ser. PPoPP '13. New York, NY, USA: ACM, 2013, pp. 289–290. [Online]. Available: http://doi.acm.org/10.1145/2442516.2442549

[6] A. Gharaibeh, L. Beltrão Costa, E. Santos-Neto, and M. Ripeanu, "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*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 345–354. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370866

[7] D. Gregor, N. Edmonds, B. Barrett, and A. Lumsdaine, "The parallel boost graph library," http://www.osl.iu.edu/research/pbgl, 2005.

[8] Y. Guo and A. Iosup, "The game trace archive," in *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 4:1–4:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2501560.2501566

[9] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, "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*, ser. ASPLOS XVII. New York, NY, USA: ACM, 2012, pp. 349–362. [Online]. Available: http://doi.acm.org/10.1145/2150976.2151013

[10] S. Hong, S. Depner, T. Manhardt, J. Van Der Lugt, M. Verstraaten, and H. Chafi, "PGX.D: A fast distributed graph processing engine," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 58:1–58:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807620

[11] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. P. PÃľrez, T. Manhardt, H. Chafi, M. Capota, N. Sundaram, M. Anderson, I. G. Tanase, Y. Xia, L. Nai, and P. Boncz, "Ldbc graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms, a technical report," Delft University of Technology, Tech. Rep., 2016.

[12] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ser. ICDM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 229–238. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2009.14

[13] A. Kyrola, G. Blelloch, and C. Guestrin, "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)*. Hollywood, CA: USENIX, 2012, pp. 31–46. [Online]. Available: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola

[14] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, pp. 985–1042, Mar. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1756039

[15] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Arg, "Introducing the graph500," Cray User's Group, Tech. Rep., 2010.

[16] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "GraphBIG: Understanding graph computing in the context of industrial solutions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 69:1–69:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807626

[17] W. L. Ngai, "Fine-grained performance evaluation of large-scale graph processing systems," Master's thesis, Delft University of Technology, 2015.

[18] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proceedings of ACM Symposium on Operating Systems Principles*, ser. SOSP '13, 2013, pp. 456–471. [Online]. Available: http://iss.ices.utexas.edu/Publications/Papers/nguyen13.pdf

[19] Y. Perez, R. Sosič, A. Banerjee, R. Puttagunta, M. Raison, P. Shah, and J. Leskovec, "Ringo: Interactive graph analytics on big-memory machines," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1105–1110. [Online]. Available: http://doi.acm.org/10.1145/2723372.2735369

[20] M. A. Rodriguez, "The gremlin graph traversal machine and language," *CoRR*, vol. abs/1508.03843, 2015. [Online]. Available: http://arxiv.org/abs/1508.03843

[21] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *SIGPLAN Not.*, vol. 48, no. 8, pp. 135–146, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2517327.2442530

[22] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey, "Graphmat: High performance graph analytics made productive," *Proc. VLDB Endow.*, vol. 8, no. 11, pp. 1214–1225, jul 2015. [Online]. Available: http://dx.doi.org/10.14778/2809974.2809983

[23] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management*

Experiences and Systems, ser. GRADES '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2484425.2484427

[24] J. Zhong and B. He, "Medusa: Simplified graph processing on gpus," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1543–1552, Jun. 2014. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2013.111