A Comparison of Parallel Graph Processing Implementations

Samuel D. Pollard
Computer and Information Science
University of Oregon
Eugene, OR 97403
Email: spollard@cs.uoregon.edu

Boyana Norris
Computer and Information Science
University of Oregon
Eugene, OR 97403
Email: norris@cs.uoregon.edu

***Abstract*—The rapidly growing number 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. Determining the best approach for a given problem is infeasible for most developers. 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 processing packages: GraphMat, Graph500, Graph Algorithm Platform Benchmark Suite, 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.**

I. Extended Abstract

A. Motivation

Any user wishing to perform graph analytics faces the daunting task of selecting which software package to use given a problem. Installing, satisfying dependencies, and determining which algorithms are supported is nontrivial. Input file formats, and stopping criteria vary across packages.

B. Architectural Overview

Our framework, easy-parallel-graph-$*$, comprises Bash shell scripts which automate each step of the experiment. Our framework breaks the process of characterizing performance into five principal phases as follows.

1) Installing modified, stable forks of each software package to ensure homogeneity.
2) Given a synthetic graph size or a real-world graph file, generate the files necessary to run each software package.
3) Given a graph and the number of threads, run each algorithm using each software package multiple times.
4) Parse through the log files to compress the output into a CSV.
5) Analyze the data using the provided R scripts to generate plots.

The source code is freely available at https://github.com/HPCL/easy-parallel-graph.

C. Algorithms and Packges

We provide analysis for three algorithms and five packages, though not all packages implement all algorithms. The algorithms are Breadth First Search (BFS), Single Source Shortest Paths (SSSP), and PageRank. The packages are listed below:

1) The Graph500 [9] consists of a specification and reference implementation of BFS.
2) The Graph Algorithm Platform (GAP) Benchmark Suite [1], a set of reference implementations for shared memory graph processing. GAP implements all three algorithms.
3) GraphBIG [10] benchmark suite, all three algorithms.
4) GraphMat [12], a library and programming model, all three algorithms.
5) PowerGraph [4], a library for distributed and shared memory graph-parallel computation. Powergraph provides SSSP and PageRank reference implementations.

D. Related Work

Graphalytics [3] is the most prominent benchmark suite presented and is still active. Other benchmark suites which to the best of our knowledge do not have associated publications are GraphBench[1] and Graph Package Testing[2]. Additionally, each graph processing package typically presents its own performance analysis.

E. Datasets

Our framework supports synthetic datasets consisting of Kronecker Graphs [7], a generalization of the RMAT graphs and are used in the Graph500.

Additionally, any dataset in the format of the Stanford Network Analysis Project [8] (SNAP) may be used[3]. For our examples we use the Dota-League dataset [5] and the cit-Patents dataset [6].

Each graph is made undirected and unweighted, then converted to the formats necessary for each implementations

[1]https://github.com/uwsampa/graphbench
[2]https://github.com/robmccoll/graphdb-testing
[3]This format is one edge per line with # as comment lines.

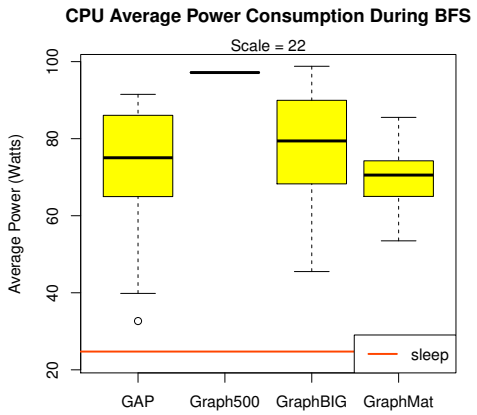

Fig. 1. We plot CPU Power Consumption for each of the 32 roots. Since Graph500 runs multiple roots per execution, we only get a single data point. The baseline monitors power consumption during the execution of the C `unistd` function `sleep(10)` (ten seconds).

F. Results

We use the Performance Application Programming Interface (PAPI) [2] to access Intel's Running Average Power Limit (RAPL), which provides a set of hardware counters for measuring energy usage. Results from this are shown in Fig. 1.

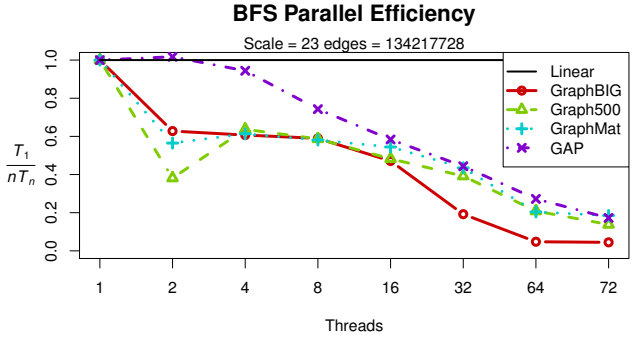

Fig. 2. Parallel efficiency for a scale-23 graph. T_1 is the serial time, n is the number of threads, and T_n is the time with n threads.

Figure 2 shows the parallel efficiency, $T_1/(nT_n)$ for different implementations of BFS. Ideal efficiency is defined as $T_n = T_1/n$ and is the horizontal line near the top of Fig. 2.

Fig. 3 shows results for real-world experiments.

G. Conclusion and Future Work

A comparison of implementations requires tedious engineering effort. The newest framework, GAP, is the best performing framework in most cases.

Future improvements will add more algorithms and packages; support for triangle counting and Galois [11] is nearly complete.

Searching for optimal parameters for SSSP (Δ-stepping) and BFS (direction-optimizing parameters α and β) via search would also increase performance for graph package users.

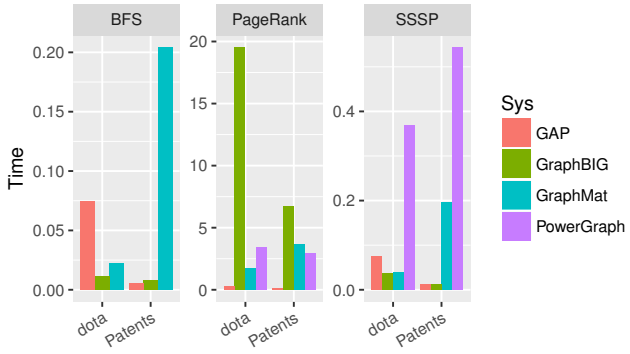

Fig. 3. Real world experiments using *easy-parallel-graph-*$*$ averaged across 32 runs.

References

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

[2] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 189–204, Aug. 2000.

[3] 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 Graph Data Management Experiences and Systems*, ser. GRADES '15. New York, NY, USA: ACM, 2015, pp. 7:1–7:6.

[4] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, ser. ODSI '12. Hollywood, CA: USENIX, 2012, pp. 17–30.

[5] 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.

[6] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *International Conference on Knowledge Discovery and Data Mining*, ser. SIGKDD. ACM, 2005.

[7] 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.

[8] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," , Jun. 2014.

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

[10] 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.

[11] 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.

[12] 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," *The Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1214–1225, jul 2015.