SCIENCE CHINA
Information Sciences

• LETTER •

December 2020, Vol. 63 229101:1–229101:3
https://doi.org/10.1007/s11432-019-2807-4

Demystifying graph processing frameworks and benchmarks

Junyong DENG[1,2*†], Qinzhe WU[2†], Xiaoyan WU[3], Shuang SONG[2], Joseph DEAN[2] & Lizy Kurian JOHN[2]

[1]*School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an* 710121*, China;*
[2]*Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin* 78712*, USA;*
[3]*School of Software Engineering, Xi'an Jiaotong University, Xi'an* 710049*, China*

Received 13 August 2019/Revised 23 November 2019/Accepted 16 February 2020/Published online 30 June 2020

Citation Deng J Y, Wu Q Z, Wu X Y, et al. Demystifying graph processing frameworks and benchmarks. Sci China Inf Sci, 2020, 63(12): 229101, https://doi.org/10.1007/s11432-019-2807-4

Dear editor,

Graph algorithms have become important because of the increasing need to extract information from big data [1]. Due to huge volumes of data and irregular communication patterns [2], it is a challenging task to apply graph algorithms efficiently. Considerable efforts have been made to improve the performance of graph processing using novel hardware designs and to facilitate application development by implementing various frameworks. Introduction of such frameworks has led to the fact that many applications are implemented through different existing versions. For instance, within the four frameworks/benchmark suites (i.e., GraphMat, Graph Algorithm Platform (GAP), GraphBIG, and Graph500) investigated in the study, there are already six implementations of the breadth first search (BFS), six implementations of the single source shortest path (SSSP) algorithm, five implementations of triangle counting (TC), four implementations of PageRank (PR), and many other applications in common. Generally, the characteristics of various implementations are not always completely clear to the research and design community.

From the hardware perspective, benchmarks are of great importance for evaluation of various proposed architectures. However, different implementations of a single graph algorithm can vary significantly in terms of scalability, computational operations, data movement, energy consumption and so on. Such variations increase the difficulty associated with defining benchmarks for these applications, which is required for the future architecture design. Therefore, it is critically important for hardware researchers to have detailed information about the characteristics of various implementations corresponding to the particular graph applications. To address this problem, we evaluate the mainstream graph applications such as BFS, SSSP, PR, and TC with respect to the four graph processing frameworks/benchmarks.

Methodology. This study is conducted using the servers of the Texas Advanced Computing Center (TACC) equipped with Intel Xeon Platinum 8160 (Skylake) processors.

The graph datasets used for the analysis are the six real world graphs and one synthetic graph including two broad categories: meshes and social networks with power-law distribution.

We utilize the profiling tool perf to obtain the architectural statistics. The compiler configurations are provided together with the source code or suggested by the technical documents of the considered graph frameworks/benchmarks.

In this study, we define several performance metrics to conduct the comparison between different frameworks, including data movement per

* Corresponding author (email: djy@xupt.edu.cn)
† Junyong DENG and Qinzhe WU have the same contribution to this work.

edge, IPC (instructions per cycle), MPKI (misses per Kilo instructions), computation per edge, execution time, and energy consumption per edge.

Appendix A presents details about conducting the experiments and corresponding setups.

Observations. Based on comparison of the four frameworks/benchmarks, we present the observations from multiple perspectives aiming to provide insights to identify appropriate benchmarks.

With respect to micro-architecture metrics, architects have developed efficient micro-architecture techniques and achieved great advancement in this field. The results of the conducted experiments on a wide range of graph processing frameworks/benchmarks and applications indicate that metrics such as IPC and MPKI may be misleading in particular cases. Kiviat diagrams presented in Figures S2(20) and S2(18) in Appendix display the eight metrics (execution yime per ddge, computation per edge, data movement per edge, IPC, L1D MPKI, L2 MPKI, L3 MPKI, and energy consumption per edge) from the top in clockwise order by the normalized values presented in the logarithmic scale. The performance of GraphBIG, as represented in Figure S2(20), is related with the understanding that the largest cache MPKIs and smallest IPC lead to the longest execution time per edge. Figure S2(20) represents the intuitive observation that with a decrease in data movement, energy consumption declines as well. Among the three implementations of TC, GAP demonstrates the best performance that is partially owing to its IPC and cache MPKIs. In turn, GraphMat is characterized with competitive IPC and even lower cache MPKIs; however, it takes around 60× execution time compared with GAP, as GAP has only 0.28% data movement per edge and therefore, saves considerable resources with respect to computation per edge as well. Figure S2(18) outlines the significance of considering data movement per edge in graph workloads. The ranking of the three frameworks/benchmark suites is reversed in SSSP. It should be noted that execution time is correlated with data movement per edge. Although GraphMat attains much larger IPC and much smaller cache MPKIs, it is not as fast as GraphBIG, as GraphMat stores graphs in the form of matrices, and uses considerably larger instructions count to accomplish the same task.

Scalability is considered as a significant property of graph processing systems, as (1) a graph might be too large for a single CPU core to process; (2) most of the capability improvements become possible owing to increasing core counts; and (3) there exists considerable potential parallelism in graph analytics workloads that needs to be explored. We run the graph frameworks/benchmarks using different number of threads (namely, 1, 4, 8, 24, 48), and extract the information useful for developing scalable parallel graph processing systems.

Figures 1(a)–(d) represent the four selected test cases in two types of setup corresponding to the execution time on the different number of cores. One type of setup implies using cross-socket communication, and the other one does not. Figure 1(a) and (b) represent execution time of BFS on OK. It can be clearly seen that GraphBIG has the inflection point on 24-core setup in Figure 1(a), but not in Figure 1(b). However, the values of the absolute execution times corresponding to GraphBIG's BFS on OK (Orkut) in 4-core, 8-core, and 24-core setups, as can be observed in Figure 1(a), are much smaller than that of the same setup presented in Figure 1(b). There might be other types of overhead, such as those caused by creating and synchronizing more threads; however, cross-socket communication is associated with the main cost from the hardware perspective. Therefore, we can conclude that the execution time can be decreased further using a larger number of cores, but with overhead caused by cross-socket communication. Nevertheless, it is possible to overcome this negative effect and benefit from adding more cores. The matrix representation used in GraphMat to construct graphs allows GraphMat being less affected by overheads. This reveals the importance of memory layout.

Another observation is that regardless of the setup, GAP BFS demonstrates the shortest execution time. However, as shown in Figures 1(c) and (d), the performance of GAP SSSP deteriorates as the number of threads increases, specifically on road networks. This is because the δ-stepping algorithm in GAP SSSP relies on the diversity of vertices. By contrast, it should be noted that the shortest execution time observed in Figures 1(c) and (d) corresponds to GraphBIG on the 1-core setup, meaning that a single-thread execution surpasses all other multi-thread executions. By inspecting the GraphBIG's source code, we have found that GraphBIG actually has a dedicated code path for its single-thread execution. That is also the reason why we do not draw a line between 1-core and 4-core setups of GraphBIG. Considering the limited parallelism in low-degree mesh graphs, the code exclusively optimized for the single-thread execution may demonstrate the superior performance compared to multi-core implementations.

The data movement per edge indicates the following three reasons of differences: applications,

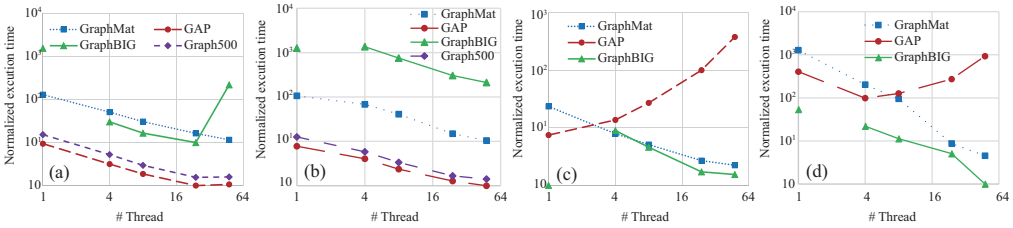

Figure 1 (Color online) (a) and (b), (c) and (d) are the values of execution time. (a) BFS OK (w/o cross-socket communication); (b) BFS OK (with cross-socket communication); (c) SSSP CA (w/o cross-socket communication); (d) SSSP CA (with cross-socket communication).

graphs, and frameworks/benchmark suites. We explain them in more detail below.

• **Applications.** Similarly as in the most simplified applications, as well as the "skeleton" for many other graph applications, on average, BFS is associated with significantly less data movement compared with the other three considered algorithms. In turn, PR is characterized with the least variation among implementations/inputs compared to the other three applications. The bar group of PR shows relatively greater flatness compared with others. This can be partially explained by the fact that PR always accesses all vertices in each iteration, while other applications have a frontier, which varies from iteration to iteration.

• **Graphs.** It is evident that graph topology affects data movement. CA (RoadCA) and TX (RoadTX) are two mesh networks with the low degree and high diameter compared to other considered graphs. This leads to deeper traversal in SSSP and BFS. It is more likely for CA and TX to move the large amount of data in BFS and SSSP. This does not hold for TC, as it does not necessitate such traversal scheme.

• **Frameworks/benchmark suites.** In the data movement chart, all groups of bars correspond to implementations of different frameworks/benchmarks for a given task. In general, there are considerable differences in results among the considered frameworks. The differences are highest in TC on road network.

It should be noted that the variations between different frameworks/benchmarks become larger with a greater extent of the variations between different applications or graphs. There are several cases in which different implementations have difference in crossing two grids (100×), and it is seen that GraphMat requires even 358× more movements than GAP to complete the same task. Hardware and software designers may consider this observation in different ways. For architects, it is not recommended to implement graph processing applications as benchmarks by themselves; otherwise it is difficult to identify whether an improvement comes from a new hardware architecture or from an implementation difference. For programmers, it is recommended to be aware of how easily redundant data movement can be introduced. Therefore, it is important to manage the memory efficiently and exploit the locality.

Based on the aforementioned analysis results, we propose several recommendations on defining graph workloads as benchmarks, as presented in Appendix G; moreover, we provide the experimental data to derive the conclusion.

Conclusion. At present, there are various frameworks and benchmark suites designed for graph processing. In this study, we observe that different available and widely used implementations of the same graph application can have distinct performance characteristics even within the same graph data set. We consider that the derived observations, as well as the obtained data statistics (including Kiviat diagrams and scalability charts) provided in Appendix contribute to the knowledge on graph processing benchmarks, by enabling both system and hardware researchers to select appropriate benchmarks in future design evaluations.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61602377, 61834005), National Natural Science Foundation of USA (Grant Nos. 1745813, 1725743), and International Science and Technology Cooperation Program of Shaanxi (Grant No. 2018KW-006).

Supporting information Appendixes A–G. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Gui C Y, Zheng L, He B, et al. A survey on graph processing accelerators: challenges and opportunities. J Comput Sci Technol, 2019, 34: 339–371
2 Song S, Liu X, Wu Q Z, et al. Start late or finish early: a distributed graph processing system with redundancy reduction. Proc VLDB Endow, 2018, 12: 154–168