RealGraph$^{\text{GPU}}$: A High-Performance GPU-Based Graph Engine toward Large-Scale Real-World Network Analysis

Myung-Hwan Jang
Hanyang University
Seoul, Republic of Korea
sugichiin@hanyang.ac.kr

Yunyong Ko
Hanyang University
Seoul, Republic of Korea
koyunyong@hanyang.ac.kr

Dongkyu Jeong
Hanyang University
Seoul, Republic of Korea
dkjeong91@hanyang.ac.kr

Jeong-Min Park
Hanyang University
Seoul, Republic of Korea
jmpark96@hanyang.ac.kr

Sang-Wook Kim*
Hanyang University
Seoul, Republic of Korea
wook@hanyang.ac.kr

ABSTRACT

A graph, consisting of vertices and edges, has been widely adopted for network analysis. Recently, with the increasing size of real-world networks, many graph engines have been studied to efficiently process large-scale real-world graphs. RealGraph, one of the state-of-the-art single-machine-based graph engines, efficiently processes storage-to-memory I/Os by considering unique characteristics of real-world graphs. Via an in-depth analysis of RealGraph, however, we found that there is still a chance for more performance improvement in the computation part of RealGraph despite its great I/O processing ability. Motivated by this, in this paper, we propose RealGraph$^{\text{GPU}}$, a GPU-based single-machine graph engine. We design the core components required for GPU-based graph processing and incorporate them into the architecture of RealGraph. Further, we propose two optimizations that successfully address the technical issues that could cause the performance degradation in the GPU-based graph engine: buffer pre-checking and edge-based workload allocation strategies. Through extensive evaluation with 6 real-world datasets, we demonstrate that (1) RealGraph$^{\text{GPU}}$ improves RealGraph by up to 546%, (2) RealGraph$^{\text{GPU}}$ outperforms existing state-of-the-art graph engines *dramatically*, and (3) the optimizations are *all* effective in large-scale graph processing.

CCS CONCEPTS

• **Information systems → Data management systems**.

KEYWORDS

graph engine; large-scale graphs processing; single machine

ACM Reference Format:
Myung-Hwan Jang, Yunyong Ko, Dongkyu Jeong, Jeong-Min Park, and Sang-Wook Kim. 2022. RealGraph$^{\text{GPU}}$: A High-Performance GPU-Based Graph Engine toward Large-Scale Real-World Network Analysis. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557679

*Corresponding author.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557679

1 INTRODUCTION

In real-world networks, there are many types of objects, which have complex relationships with each other [30, 32]. By analyzing such a network, we can obtain useful information and knowledge to leverage them in various downstream tasks such as community detection, link prediction, and recommendation [7, 10, 22, 28]. Recently, with the increasing size of real-world networks (e.g., social networks), *graph engines* for efficiently analyzing large-scale real-world networks have been widely studied [6, 8, 15, 18, 19, 29, 33, 35]. In *single-machine-based graph engines* [6, 8, 15, 33, 35], they store an entire graph exceeding the main memory (MM) capacity in external storage (e.g., HDD and SSD) and load *parts* of the graph (i.e., vertices and their related edges) into MM only when they are required for processing. In this way, they successfully process large-scale graphs on a single machine, while showing great performance comparable to or even better than *distributed-system-based graph engines*, with limited computing resources. Although the single-machine-based approach is not able to process extremely large graphs exceeding the capacity of external storage, it is very useful in practice because not only most real-world graphs fit in external storage but also it does not require expensive infrastructure.

The authors of RealGraph [8], the state-of-the-art single-machine-based graph engine, identified the unique characteristics of real-world graphs, and proposed a novel 4-layer architecture and optimizations to reflect those characteristics. Thanks to its inherent architecture and optimizations, RealGraph efficiently addresses *storage-to-MM I/Os*, the main challenge of single-machine-based graph engines, thereby improving significantly the graph processing performance compared to existing graph engines [6, 15, 24, 33, 35]. For an in-depth analysis, we performed four popular graph algorithms having different patterns on Twitter and Yahoo datasets with RealGraph and measured the computation and I/O overheads. Figure 1 shows the *ratio* of computation and I/O overheads of RealGraph. Clearly, the I/O overhead is always much lower than the computation overhead across all graph algorithms. This indicates that RealGraph successfully handles storage-to-MM I/Os occurred inevitably by a limited memory size on a single machine.

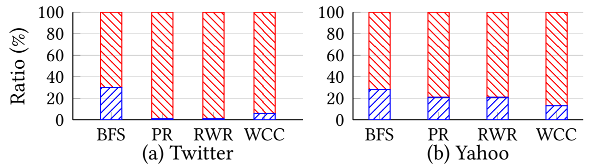

Figure 1: The ratio of computation (red) and I/O (blue) overheads of RealGraph [8] on two real-world datasets.

Under this circumstance, in this work, we aim to further improve the performance of RealGraph. To this end, we look into common characteristics of graph algorithms closely. Most graph algorithms consist of *repeated independent operations* for vertices and edges [12, 25, 26]. In the case of PageRank [21], each vertex sends its PageRank score to its out-neighbors and aggregates the scores received from its in-neighbors at every iteration. Here, the operation for each vertex is *independent* of those for other vertices. This naturally indicates that the operations of graph algorithms can be *parallelized.* Meanwhile, a GPU is a computational accelerator composed of hundreds/thousands of cores specially designed for fast parallel computing [11, 20, 27]. Thus, a GPU is a much more-suitable device than a CPU in processing large-scale graphs due to its strong parallel computing power. Since RealGraph is a CPU-based engine, despite its great I/O capability, there is still a chance for more improvement in the computation part (as shown in Figure 1).

This motivates us to propose RealGraph^{GPU}, a GPU-based single-machine graph engine. To this end, we design new components required for efficient GPU-based graph processing, incorporate them into the architecture of RealGraph, and construct a novel 5-layer architecture (Section 3.1). Based on the architecture, RealGraph^{GPU} loads only the necessary parts of a graph into the GPU device memory (DM) and processes them in parallel by a number of GPU cores. Further, we identify technical issues that can cause significant performance degradation in GPU-based graph processing and propose two novel optimization strategies for addressing them effectively: (1) the buffer pre-checking to reduce the amount of unnecessary I/Os, and (2) the edge-based workload allocation to distribute workloads to GPU threads *evenly.* As a result, RealGraph^{GPU} can process large-scale graphs very efficiently by leveraging strong parallel-computing power of a GPU, while maintaining the efficient I/O processing power which is the original strength of RealGraph.

We validate the superiority of our RealGraph^{GPU} in comparison with *six* state-of-the-art graph engines, including RealGraph, by performing extensive experiments with *four* popular graph algorithms on *six* real-world graphs. The experimental results demonstrate that (1) RealGraph^{GPU} improves the performance of RealGraph significantly by up to 546%; (2) RealGraph^{GPU} outperforms all the state-of-the-art graph engines *dramatically* by up to 70 times; (3) our optimization strategies employed in RealGraph^{GPU} are all beneficial to large-scale graph processing.

2 RELATED WORKS

Single-machine-based approach. Single-machine-based graph engines [3, 6, 8, 15, 24, 35] focus on efficiently handling storage-to-MM I/Os, the main challenge of the single-machine-based approach. GraphChi [15] and X-Stream [24] improve the I/O processing performance by exploiting the *sequential access* to data in storage, rather than random access. TurboGraph [6], GridGraph [35], and FlashGraph [33] aim to reduce unnecessary storage-to-MM I/Os and utilize the parallel I/O processing ability of SSD. RealGraph [8], the state-of-the-art graph engine, analyzes the characteristics of real-world graphs and the operation patterns of graph algorithms, thereby achieving high performance in storage-to-MM I/Os.

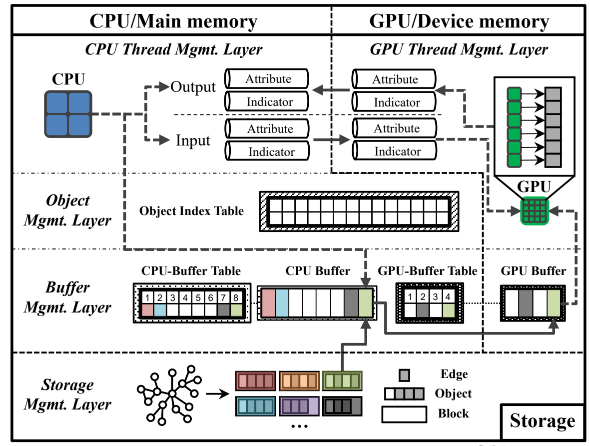

Figure 2: Architecture of RealGraph^{GPU}.

Distributed-system-based approach. Many distributed-system-based graph engines have been widely studied as well [2, 18, 19, 23, 29, 34]. The graph engines belonging to this approach split and distribute the entire graph over multiple nodes in a distributed system, and process them in parallel. In this way, this approach is able to process extremely large graphs that the single-machine-based approach fails to process. However, the distributed-system-based approach requires not only inter-node communication, which is very time-consuming, but also costly infrastructure to support the inter-node communication [1, 5, 13, 17, 34].

Relation to our work. Most existing graph engines focus mainly on improving the performance of I/O processing, while little attention has been paid to improving the computation performance. Our work aims at improving the computation performance in graph processing by leveraging the strong parallel computing power of a GPU on a single machine, while maintaining the efficient I/O processing power of the original RealGraph.

3 PROPOSED METHOD: REALGRAPH^{GPU}

3.1 Architecture and Algorithm

3.1.1 Architecture. RealGraph [8] proposes a novel 4-layer architecture (i.e., storage, buffer, object, and CPU thread management layers), where each layer closely interacts with the other layers to process storage-to-MM I/Os efficiently. Toward extending RealGraph to a GPU-based graph engine, in this work, we (1) design a new layer (GPU thread management layer) to manage GPU threads and device memory (DM), and (2) refine the original buffer and CPU thread management layer to support efficient GPU-based graph processing. Figure 2 illustrates the 5-layer architecture and the processing flow of RealGraph^{GPU}. The description of each layer is as follows.

- **Storage management layer:** This layer manages the data stored in storage, where data is stored in fixed-size blocks and processed in a block-based manner. Each block includes multiple objects, each of which stores a vertex and its related edges.
- **Buffer management layer:** This layer is in charge of the blocks that are loaded in the CPU and GPU buffers (i.e., the blocks in

MM and DM). It manages the CPU and GPU buffers for physically storing blocks in MM and DM, and the CPU/GPU-buffer tables for indexing the loaded blocks.

- **Object management layer:** This layer manages the information about which objects are included in each block. In the object index table, the i^{th} column represents the indices of the objects in the i^{th} block, where each object is indexed with its corresponding vertex ID. To reduce the main memory consumption, we only store the first and last objects' indices (i.e., the two vertex IDs) after sorting the objects in ascending order.
- **CPU thread management layer:** This layer manages CPU threads and attribute/indicator vectors in MM. CPU threads are in charge of the MM-to-DM data transfer (i.e., sending/receiving the data to/from GPU DM). The attribute vectors store the result (e.g., PageRank scores) from the current/next iterations and the indicator vectors store the information about which vertices to be processed in the current/next iteration.
- **GPU thread management layer:** This layer manages GPU threads and attribute/indicator vectors in DM. GPU threads perform the actual operations of a graph algorithm (e.g., PageRank) based on the attribute/indicator vectors. The attribute/indicator vectors play the same roles as in the CPU management layer. After the operations are completed, the results are transferred back to the CPU thread management layer.

3.1.2 Algorithm. Since the GPU DM is limited, RealGraph$^{\text{GPU}}$ loads only necessary parts of a graph into DM and processes them in parallel by using GPU threads, based on the 5-layer architecture. Algorithm 1 shows the entire process of RealGraph$^{\text{GPU}}$, where B_i indicates the i^{th} block in the external storage, T_{obj} does the object index table, $T_{\text{cpu}}/T_{\text{gpu}}$ do the CPU/GPU-buffer tables, $f(\cdot)$ does the function of a given graph algorithm, and $a_{\text{in/out}}$ and $d_{\text{cnt/next}}$ do the attribute and indicator vectors, respectively. At each iteration, RealGraph$^{\text{GPU}}$ loads the block (B_i) having the vertices and their related edges to be processed into the GPU buffer and runs the operations of a given graph algorithm ($f(\cdot)$) (lines 3-7 in Algorithm 1).

Algorithm 1 Graph processing of RealGraph$^{\text{GPU}}$

1: **Function** RealGraph$^{\text{GPU}}$(B, T_{obj}, f):
2: $a_{\text{in}}, a_{\text{out}}, d_{\text{cnt}}, d_{\text{next}} \leftarrow 0, T_{\text{cpu}}, T_{\text{gpu}} \leftarrow \emptyset$
3: **for** $t = 0, 1, \ldots$ **do**
4: $B_i \leftarrow$ get_next_block($B, T_{\text{obj}}, T_{\text{gpu}}, T_{\text{cpu}}, d_{\text{cnt}}$)
5: $a_{\text{out}}, d_{\text{next}} \leftarrow f(B_i, a_{\text{in}}, d_{\text{cnt}})$ *# run a graph operation*
6: $a_{\text{in}} \leftarrow a_{\text{out}}, d_{\text{cnt}} \leftarrow d_{\text{next}}$
7: **end for**
8: **return** a_{out}

9: **Function** get_next_block($B, T_{\text{obj}}, T_{\text{gpu}}, T_{\text{cpu}}, d_{\text{cnt}}$):
10: $i \leftarrow$ get_object_index($T_{\text{obj}}, d_{\text{cnt}}$)
11: **if** $B_i \notin T_{\text{gpu}}$ **then** *# buffer pre-checking*
12: **if** $B_i \notin T_{\text{cpu}}$ **then** *# Storage-to-MM-to-DM*
13: $T_{\text{gpu}} \leftarrow \{B_i\} \cup T_{\text{gpu}}, T_{\text{cpu}} \leftarrow \{B_i\} \cup T_{\text{cpu}}$
14: **else** *# MM-to-DM*
15: $T_{\text{gpu}} \leftarrow \{B_i\} \cup T_{\text{gpu}}$
16: **end if**
17: **end if**
18: **return** B_i

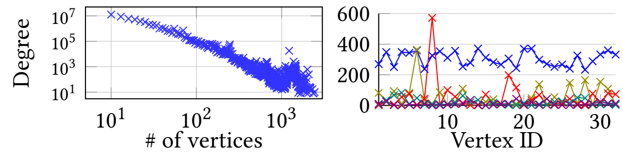

Figure 3: The power-low degree distribution (left) and the different degrees across vertices and blocks (right).

3.2 Performance Optimizations

3.2.1 Buffer pre-checking. The operations of a graph algorithm are repeated for vertices (or edges). In general, the vertex processed in the previous iteration and its neighbors tend to be processed *again* in the next iteration [9]. If a graph engine does not take into account which vertices and edges were processed in the previous iteration, it tries to transfer the data to the GPU buffer at every iteration because it is unaware of which vertices and edges are currently in the GPU buffer. That is, even though the vertices needed in the current iteration are already loaded in the CPU/GPU buffers, the block storing these vertices should be transferred *unnecessarily*, which may cause serious performance degradation.

To address this issue, we define the CPU/GPU-buffer tables managing the indices of the blocks loaded in the CPU/GPU buffer, and propose a simple yet effective strategy to reduce the unnecessary MM-to-DM and storage-to-MM I/Os (**buffer pre-checking**). Function get_next_block($\cdot$) in Algorithm 1 describes the block loading process of RealGraph$^{\text{GPU}}$ with the buffer pre-checking strategy. RealGraph$^{\text{GPU}}$ checks the GPU/CPU-buffer tables at the beginning of each iteration, for deciding whether to request data to the CPU thread/storage management layers (lines 11-17). Additionally, we implement the MM-to-DM data transfer using *asynchronous streams* supported by the GPU [4], thereby hiding the overhead of the MM-to-DM data transfer under the GPU processing overhead.

3.2.2 Edge-based workload allocation. In general, real-world graphs tend to follow a *power-law* degree distribution, which means that a majority of vertices have a small number of edges while a small number of vertices have a huge number of edges [14, 16]. Figure 3 clearly shows that a real-world graph follows the power-law degree distribution (left) and the degrees of vertices are quite different across blocks (right). This implies that *the amount of required operations is also quite different across vertices* in real-world graphs. The vertex-based workload allocation that many graph engines [6, 15, 24, 33] have adopted, however, distributes workloads into multiple threads in a vertex-based manner, without taking into account this unique characteristic of real-world graphs. As a result, a few threads in charge of the vertices with a huge number of edges could be overloaded, thus degrading the entire performance significantly for large-scale real-world graph processing.

From this observation, we propose an **edge-based workload allocation** strategy that aims to distribute the workloads into GPU threads *evenly*. RealGraph$^{\text{GPU}}$ with the edge-based workload allocation distributes workloads into GPU threads in an edge-based manner (rather than the vertex-based one) and run the same graph operation on multiple GPU threads in parallel (line 5). Through our edge-based workload allocation, RealGraph$^{\text{GPU}}$ is able to *balance well* the workloads over GPU threads regardless of the vertices having a large number of edges. Note that the edge-based strategy is always superior to the vertex-based one in balancing workloads

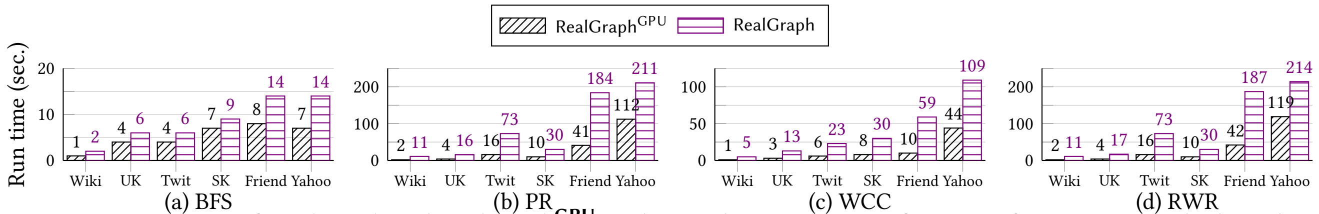

Figure 4: Comparison of RealGraph and RealGraph^{GPU} in the graph processing performance for various graph algorithms.

across GPU threads, except for the extreme case where all vertices have the exactly same number of edges. We empirically verify the effectiveness of the above two optimization strategies in Section 4.

4 EVALUATION

In this section, we evaluate RealGraph^{GPU} with real-world datasets by answering the following evaluation questions (EQs):

- EQ1: How much does RealGraph^{GPU} improve RealGraph in terms of the performance of graph processing?
- EQ2: Does RealGraph^{GPU} provide the performance better than the existing state-of-the-art graph engines?
- EQ3: Are the optimization strategies effective in improving the performance of RealGraph^{GPU}?

Experimental setup. We run our experiments on a single machine equipped with an Intel i7-8700K CPU with 128GB main memory (MM), 250GB M.2 NVMe SSD, and Titan XP GPU with 12GB device memory (DM) using PCIe Gen3 interface. We set the number of CPU threads and GPU streams as 8 and 32, respectively, and limit MM and DM as 16GB and 8GB respectively, to rigorously evaluate RealGraph^{GPU} in a limited MM and DM environment. We set the size of each block to 1MB, same as [8]. We use six real-world datasets [8] (Table 1) and four popular graph algorithms – breadth-first search (BFS) [25], PageRank (PR) [21], weakly connected component (WCC) [26], and random walk and restart (RWR) [31].

Table 1: Statistics of real-world datasets.

Datasets	Wiki	UK	Twitter	SK	Friend	Yahoo
# of Nodes	12M	39M	61M	50M	68M	1,4B
# of Edges	370M	930M	1.4B	1.9B	2.5B	6.6B
Size	5.7GB	16GB	24GB	32GB	44GB	114GB

EQs1-2. Performance of RealGraph^{GPU}. In this experiment, we compare RealGraph^{GPU} with RealGraph [8] and five existing graph engines – GraphChi [15], X-Stream [24], TurboGraph [6], GridGraph [35], and FlashGraph [33]. We run the four graph algorithms on six real-world graphs by using each graph engine, and measure the running time. First, Figure 4 shows that *RealGraph^{GPU} always outperforms RealGraph across all graph algorithms and all datasets* (espeically, by up to 546% gain). This result verifies that the proposed architecture and optimization strategies of RealGraph^{GPU} are quite effective in extending RealGraph toward a GPU-based graph engine. Second, Figure 5 shows that RealGraph^{GPU} provides the performance much better than the existing graph engines (by up to ×69 better than TurboGraph), where "O.O.M" and "O.O.T" denote the *out-of-memory* and *out-of-time* indicating the case exceeding 24 hours, respectively. Thus, RealGraph^{GPU} efficiently processes even the huge graphs that existing graph engines fail to process. As a result, we validate that our RealGraph^{GPU} successfully leverages the strong parallel-computing power of a GPU, while maintaining the efficient I/O processing power, the original strength of RealGraph.

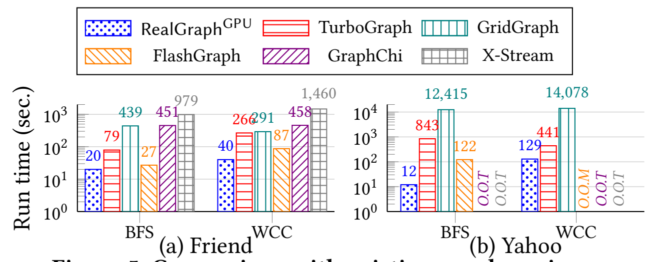

Figure 5: Comparison with existing graph engines.

EQ3. Ablation study. In this experiment, we verify the effects of our optimization strategies. We compare the following four versions of RealGraph^{GPU}: (1) RG-no is the version without any optimizations; (2) RG-Bcheck is the one with the buffer pre-checking; (3) RG-Ealloc is the one with the edge-based workload allocation; (4) RG-All is the one with all of the two strategies. We run BFS and WCC by using each of the four versions and measure the running time. Figure 6 shows the results, where the relative time of 1 in the Y-axis represents the baseline performance (RG-no). The results show that each of the proposed strategies improves the naive version of RealGraph^{GPU} (RG-no) and RG-All provides the *best* performance in all cases (by up to 960% compared to RG-no). This implies that both of the proposed strategies are effective in addressing the technical issues that we explained in Section 3.2.

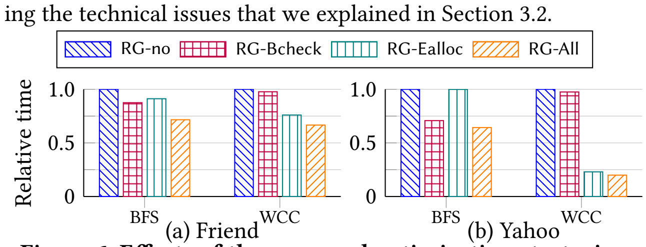

Figure 6: Effects of the proposed optimization strategies.

5 CONCLUSIONS

In this paper, we proposed a novel GPU-based single-machine graph engine, RealGraph^{GPU}, to process large-scale real-world graphs efficiently. Also, we identified two technical issues that could cause significant performance degradation and proposed novel optimization strategies for addressing them: the buffer pre-checking and the edge-based workload allocation. Via comprehensive evaluation with six real-world datasets, we showed that RealGraph^{GPU} outperforms RealGraph and existing state-of-the-art graph engines *dramatically*, and all of our optimization strategies are quite effective in improving the performance of large-scale graph processing.

6 ACKNOWLEDGMENTS

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2022-00155586 and No.2022-0-00352). Also, this is the result of the joint work with Samsung Electronics Co., Ltd.

REFERENCES

[1] Aydin Buluç and John R Gilbert. 2012. Parallel Sparse Matrix-Matrix Multiplication and Indexing: Implementation and Experiments. *SIAM Journal on Scientific Computing* 34, 4 (2012), C170–C191.

[2] Rong Chen, Jiaxin Shi, Yanzhe Chen, Binyu Zang, Haibing Guan, and Haibo Chen. 2019. Powerlyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. *ACM Transactions on Parallel Computing (TOPC)* 5, 3 (2019), 1–39.

[3] Yuze Chi, Guohao Dai, Yu Wang, Guangyu Sun, Guoliang Li, and Huazhong Yang. 2016. Nxgraph: An Efficient Graph Processing System on a Single Machine. In *Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 409–420.

[4] NVIDIA Corporation. 2022. *CUDA Stream*. https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

[5] Roshan Dathathri, Gurbinder Gill, Loc Hoang, Hoang-Vu Dang, Alex Brooks, Nikoli Dryden, Marc Snir, and Keshav Pingali. 2018. 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)*. ACM SIGPLAN, 752–768.

[6] Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, Jeong-Hoon Lee, Min-Soo Kim, Jinha Kim, and Hwanjo Yu. 2013. TurboGraph: A Fast Parallel Graph Engine Handling Billion-Scale Graphs in a Single PC. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 77–85.

[7] Min-Hee Jang, Christos Faloutsos, Sang-Wook Kim, U Kang, and Jiwoon Ha. 2016. Pin-trust: Fast Trust Propagation Exploiting Positive, Implicit, and Negative Information. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 629–638.

[8] Yong-Yeon Jo, Myung-Hwan Jang, Sang-Wook Kim, and Sunju Park. 2019. RealGraph: A Graph Engine Leveraging The Power-Law Distribution of Real-World Graphs. In *Proceedings of the 2019 World Wide Web Conference (WWW)*. ACM, 807–817.

[9] Yong-Yeon Jo, Myung-Hwan Jang, Sang-Wook Kim, and Sunju Park. 2021. A Data Layout with Good Data Locality for Single-Machine based Graph Engines. *IEEE Trans. Comput.* 14, 8 (2021), 1–10.

[10] Yoonsuk Kang, Jun-Seok Lee, Won-Yong Shin, and Sang-Wook Kim. 2022. Community Reinforcement: An Effective and Efficient Preprocessing Method For Accurate Community Detection. *Knowledge-Based Systems* 237 (2022), 107741.

[11] Farzad Khorasani, Keval Vora, Rajiv Gupta, and Laxmi N Bhuyan. 2014. CuSha: Vertex-Centric Graph Processing on GPUs. In *Proceedings of the 23rd IEEE International Symposium on High-performance Parallel and Distributed Computing (HPDC)*. IEEE, 239–252.

[12] Jon M Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632.

[13] Yunyong Ko, Kibong Choi, Jiwon Seo, and Sang-Wook Kim. 2021. An In-Depth Analysis of Distributed Training of Deep Neural Networks. In *Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 994–1003.

[14] Yunyong Ko, Jae-Seo Yu, Hong-Kyun Bae, Yongjun Park, Dongwon Lee, and Sang-Wook Kim. 2021. MASCOT: A Quantization Framework for Efficient Matrix Factorization in Recommender Systems. In *Proceedings of the 2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 290–299.

[15] Aapo Kyrola, Guy E Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 31–46.

[16] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007), 1–41.

[17] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2018. Asynchronous Decentralized Parallel Stochastic Gradient Descent. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 3043–3052.

[18] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *Proceedings of the VLDB Endowment* 5, 8 (2012), 716–727.

[19] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. ACM, 135–146.

[20] Yusuke Nagasaka, Akira Nukada, and Satoshi Matsuoka. 2017. High-Performance and Memory-Saving Sparse General Matrix-Matrix Multiplication for Nvidia Pascal GPU. In *Proceedings of the 2017 46th IEEE International Conference on Parallel Processing (ICPP)*. IEEE, 101–110.

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford InfoLab.

[22] Masoud Rehyani Hamedani and Sang-Wook Kim. 2021. AdaSim: A Recursive Similarity Measure in Graphs. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 1528–1537.

[23] Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. 2015. Chaos: Scale-out Graph Processing from Secondary Storage. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*. ACM, 410–424.

[24] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-Stream: Edge-Centric Graph Processing Using Streaming Partitions. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 472–488.

[25] Robert Sedgewick and Kevin Wayne. 2011. *Algorithms*. Addison-Wesley Professional.

[26] Kenji Suzuki, Isao Horiba, and Noboru Sugie. 2003. Linear-Time Connected-Component Labeling Based on Sequential Local Operations. *Computer Vision and Image Understanding* 89, 1 (2003), 1–23.

[27] Hao Wang, Liang Geng, Rubao Lee, Kaixi Hou, Yanfeng Zhang, and Xiaodong Zhang. 2019. SEP-Graph: Finding Shortest Execution Paths for Graph Processing under a Hybrid Framework on GPU. In *Proceedings of the 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. ACM SIGPLAN, 38–52.

[28] Hao Wang, Yan Yang, and Bing Liu. 2019. GMC: Graph-based Multi-view Clustering. *IEEE Transactions on Knowledge and Data Engineering* 32, 6 (2019), 1116–1129.

[29] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. 2013. GraphX: A Resilient Distributed Graph System on Spark. In *Proceedings of the International Workshop on Graph Data Management Experiences and Systems (GRADE)*. ACM, 1–6.

[30] Xifeng Yan, Philip S Yu, and Jiawei Han. 2004. Graph Indexing: A Frequent Structure-Based Approach. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, 335–346.

[31] Hilmi Yildirim and Mukkai S Krishnamoorthy. 2008. A Random Walk Method for Alleviating the Sparsity Problem in Collaborative Filtering. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys)*. ACM, 131–138.

[32] Peixiang Zhao and Jiawei Han. 2010. On Graph Query Optimization in Large Networks. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 340–351.

[33] Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E Priebe, and Alexander S Szalay. 2015. FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. USENIX, 45–58.

[34] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. 2016. Gemini: A {Computation-Centric} Distributed Graph Processing System. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 301–316.

[35] Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2015. GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning. In *Proceedings of the 2015 USENIX Annual Technical Conference (ATC)*. USENIX, 375–386.