Front. Comput. Sci., 2023, 17(1): 171602
https://doi.org/10.1007/s11704-022-1546-2

LETTER

A lock-free approach to parallelizing personalized PageRank computations on GPU

Zhigang WANG[1], Ning WANG (✉)[1], Jie NIE[1], Zhiqiang WEI[1], Yu GU[2], Ge YU[2]

1 Faculty of Information Science & Engineering, Ocean University of China, Qingdao 266100, China
2 School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China

1 Introduction

Personalized PageRank (PPR) is a classic topology-based proximity measure and it is most widely computed by Forward Push. That is, given a starting vertex s in graph G, it iteratively computes the importance score of any vertex u in G with respect to s, and then broadcasts the new score as messages to u's neighboring vertices. The process converges until all scores hold stable. Recently, Graphical Processing Units (GPU) with massive threads has been extensively used to parallelize such compute-intensive process. It yields performance improvement but also involves two atomic locks for correctness. Such locks are practically inefficient and become a new performance bottleneck. This paper proposes a separation technique to partially eliminate atomic protections, termed as Lightweight Forward Push. A Forward Pull solution is further devised to support lock-free PPR computations but also causes useless reads. For best performance, a new Hybrid Framework is then designed to adaptively balance locking costs and reading costs.

2 Lightweight forward push

Most PPR-related works employ an "one-pass" design because receiving a message can affect the vertex value and then make it possibly reach the update criterion θ. Such detection can be executed up to $|E|$ times because all edges might be involved in generating messages. However, as shown in Fig. 1(a), many source vertices like u_1, u_2 and u_3 might send messages to the same destination vertex like v_x concurrently. We should use heavy locks to distinguish such parallel detection so as to avoid repeatedly adding v_x into the candidate update queue. Further investigation reveals that in all $|E|$ times of detecting operations, only $|V|$ times at most are valid. Take v_x as an example. It is detected three times but only the second is valid.

Our new technique breaks such "one-pass" design. As shown in Fig. 1(b), it explicitly separates message propagation from criterion detection with a newly inserted global barrier. Vertex values keep stable after the barrier since all possible messages have been received. Then vertices can safely perform criterion detection only once to remove locks.

3 Lock-free forward pull

Next, the core design of Forward Push is that a vertex can immediately and actively deliver messages to its out-neighboring destination vertices. As a result, a destination vertex must passively and concurrently receive messages from in-neighbors across different threads, leading to heavy write races. We solve this problem by changing the message propagation policy, i.e., Forward Pull. Now a destination vertex will actively pull messages required on demand of its update. That improves the time locality of arrived messages and hence, we can allocate enough but temporary memory spaces to store collected messages, instead of atomically writing them into a single space.

Note that all of pulled messages must be sequentially accumulated into the destination vertex, which is still time-consuming. Now we try to parallelize the accumulation using binary combination. As shown in Fig. 1(c), the consecutive local storage spaces are evenly divided into two parts based on the middle position. Binary combination reads the last part values and combines them with the remaining half, which is done by threads related to the latter in parallel. Combination then continues on the remaining half. We repeat this process until all messages are accumulated into the head space.

We are aware of some existing studies also mention Pull, but they either work on CPU environments [1,2], or only support BFS-like algorithms on GPU where a vertex is updated at most once during iterations [3]. They are not suitable for PPR on GPU with massive threads and frequent updates.

4 A hybrid solution

Note that Forward Pull is still not free although it completely releases the burden of locks. This is because a destination vertex cannot exactly know which in-neighbor has been updated before sending pulling requests. It then conservatively collects messages from all in-neighbors, which clearly generates useless reads and even makes Pull underperform

Received October 5, 2021; accepted December 30, 2021

E-mail: wangning8687@ouc.edu.cn

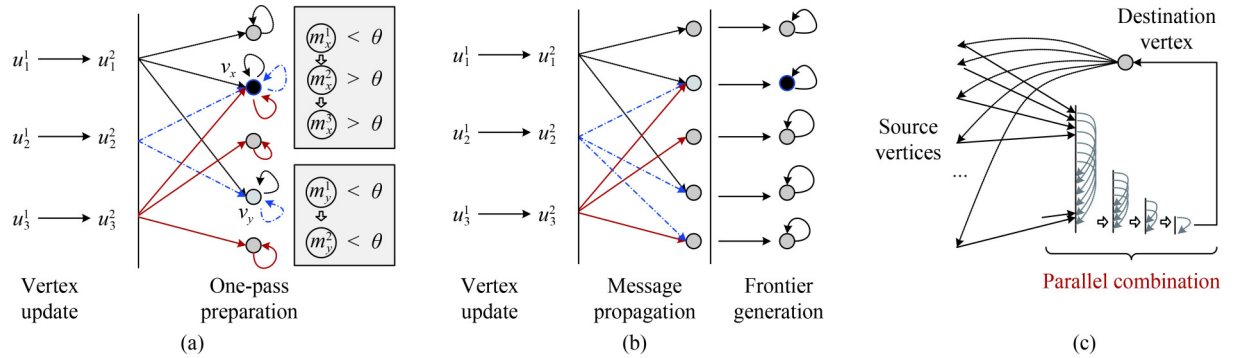

Fig. 1 Optimizations used in Forward Push and Forward Pull. (a) Existing one-pass dataflow; (b) Separated dataflow; (c) Parallel binary combination

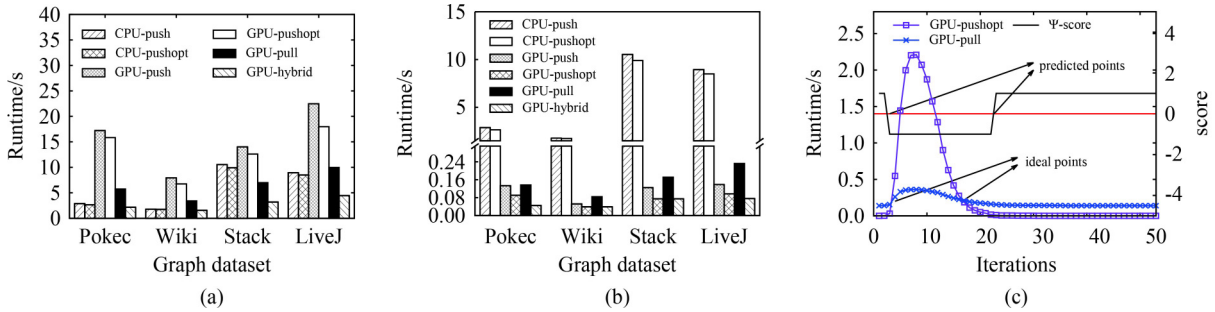

Fig. 2 Performance analysis of different solutions on CPU and GPU devices. (a) Runtime (CPU vs. *GPUL*); (b) Runtime (CPU vs. *GPUH*); (c) Effectiveness (*LiveJ* on *GPUL*)

Push if only a few in-neighbors are updated. We thereby propose a hybrid framework on top of our lightweight Push and Pull models, so that we can switch to an efficient one for a specific iteration, to balance locking costs and read costs.

We especially study the switching timing in GPU environments using the runtime difference of pushing and pulling models, denoted by Ψ. Ψ_k is computed at the end of the kth iteration and then used to predict the comparison result of the next iteration. The framework should run Push if $\Psi_k < 0$, and Pull, otherwise. For more details, please refer to the Online Resource 1.

5 Experiments

We compare our methods against the state-of-the-art GPU-based PPR solution proposed in [4], denoted by *push*. Our pushing and pulling models are respectively called as *pushopt* and *pull*. *hybrid* is used to stand for our hybrid framework.

All tests are performed on real graph datasets, including *Pokec*, *Wiki*, *Stack* and *LiveJ*. We process them on two GPU devices and a CPU device. The former two have low and high configurations, respectively denoted by GPUL and GPUH. More details are given in Online Resource 2.

Figure 2 reports experiment results, where *pushopt* and *pull* have different favorite scenarios but *hybrid* consistently performs the best. In particular, on GPUL, compared with *pushopt* and *pull*, *hybrid* respectively achieves up to 86% and 62% performance improvements (both on *Pokec*).

Figure 2(c) plots the sign of Ψ-score as a function of the iteration counter, to show switching points predicted by *hybrid*. We also manually mark the ground-truth/ideal switching points by solely running *pushopt* and *pull*. Clearly, *hybrid* can roughly select the right model.

6 Conclusion

This paper proposes a hybrid framework on top of two newly designed models to accelerate GPU-based PPR computations. Extensive experiments validate the effectiveness and efficiency.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant Nos. 61902366 and 61902365), the Fundamental Research Funds for the Central Universities (202042008), the CCF-Huawei Innovation Research Plan, the Project funded by China Postdoctoral Science Foundation (2020T130623), and the Qingdao Independent Innovation Major Project (20-3-2-12-xx).

Supporting information The supporting information is available online at journal.hep.com.cn and link.springer.com.

References

1. Shun J L, Blelloch G E. Ligra: a lightweight graph processing framework for shared memory. In: Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2013, 135–146
2. Wang Z G, Gu Y, Bao Y B, Yu G, Yu J X, Wei Z Q. HGraph: I/O-efficient distributed and iterative graph computing by hybrid pushing/pulling. IEEE Transactions on Knowledge and Data Engineering, 2021, 33(5): 1973–1987
3. Wang Y Z H, Davidson A, Pan Y C, Wu Y D, Riffel A, Owens J D. Gunrock: a high-performance graph processing library on the GPU. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2015, 265–266
4. Sha M, Li Y C, He B S, Tan K L. Accelerating dynamic graph analytics on GPUs. Proceedings of the VLDB Endowment, 2017, 11(1): 107–120