CGAcc: CSR-based Graph Traversal Accelerator on HMC

Cheng Qian, Libo Huang, QiYu, Zhiying Wang
National University of Defense Technology
ChangSha, China
{qiancheng,libohuang,yuqi13,zywang}@nudt.edu.cn

Bruce Childers
University of Pittsburgh
Pittsburgh, Pennsylvania
childers@cs.pitt.edu

ABSTRACT

Graph traversal is widely involved in lots of realistic scenarios such as road routing, social network and so on. Unfortunately graph traversal is quite time-consuming because of terrible spatial locality. Conventional prefetch technology and parallel framework do not bring much benefit. Hybrid Memory Cube (HMC) can serve as high bandwidth main memory as well as providing an enhanced logic layer with the ability of controlling memory access and processing in memory (PIM). Armed with the knowledge of graph's structure, we propose CGAcc in this paper. By deploying three prefetchers on the logic layer and making them in a pipeline way, CGAcc achieves average 2.8X speedup compared with conventional memory system with stream prefetcher.

CCS CONCEPTS

• **Hardware → Memory and dense storage**;

KEYWORDS

Hybrid Memory Cube, graph traversal, Compressed row storage,

ACM Reference Format:
Cheng Qian, Libo Huang, QiYu, Zhiying Wang and Bruce Childers. 2018. CGAcc: CSR-based Graph Traversal Accelerator on HMC. In *ICPP 2018: ICPP 2018: International Conference on Parallel Processing, August 13–16, 2018, Eugene, Oregon, USA.* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.1145/3203217.3203235

1 INTRODUCTION

Graph traversal is adopted in a wide variety of realistic scenarios, for example road routing, social relationship network, gene graph analysis and so on. Because of the terrible spatial locality, graph traversal is quite time-consuming, especially when the graph contains huge number of vertexes. Conventionally when we try to accelerate memory-bound workloads, prefetch is an efficient way to relieve the very high memory transaction latency by learning the access pattern and keeping a relative high accuracy in predict and fetch the following possible accessed data. Unfortunately, graph's access pattern is data-dependent so that very hard to predict through a fixed mode. Several classical prefetchers, such as strider prefetcher, pointer fetcher, are reported inefficient to deal with graph traversal [1] [3]. It is also quite difficult to use parallel frameworks or devices for accelerating as usually graphs cannot be obviously parallelly handled [6].

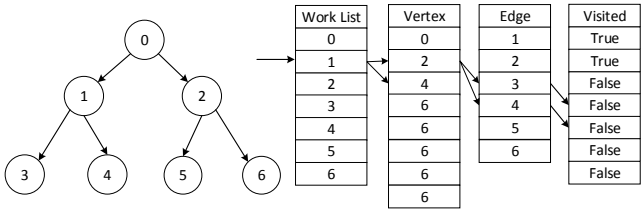

Figure 1: CSR graph traversal work flow.

However, the well-defined structure graph structure leaves chance for explicit prefetchers. Based on knowing where to locate the data used in the near future according to the knowledge of the graph structure rather than access pattern, the only problem is to accelerate this process because direct prefetch is meaningless. In this paper, we use Micron's Hybrid Memory Cube (HMC) to implement this acceleration. HMC serves as high-bandwidth main memory with an enhanced logic layer which can handle simple atomic commands as well as memory accesses. We propose **CGAcc**, a CSR based graph traversal accelerator on HMC, which deploys three prefetchers working in a pipeline style cooperatively to reduce large amounts of transaction latency. Comprehensive evaluations are concluded to show that CGAcc can achieve average 2.8X speedup.

2 ARCHITECTURE

In the purpose of reducing capacity cost, Compressed row storage (CSR) [2] is used as a representation for graph. In the presentation of CSR-based graph, three arrays (vertex, edge, visited) are needed. It is worth noted that the contents in these arrays are indexes rather than pointers. Figure 1 shows the work flow of a CSR-based graph traversal example. The index of a work node leads to the corresponding two locations (index and index + 1) in vertex array. These two values which fetched from vertex arrays illustrate the range that the data should take from edge array. Similarly, the edge data will also be used as the index for the visited array. Finally, the visited array will be accessed to determine whether this extended node has been accessed or not, and if not, this node will be pushed into the work list as a new node.

The work flow shown in Figure 1 implies the possibility for pipelining it. As Figure 2 illustrates, CGAcc deploys three prefetchers on the logic layer of HMC. These prefetchers work in a pipeline style cooperatively. In our mechanism CGAcc acts like a master rather than a slave, which means what CPU side needs to do is only send a start request. After the request is accepted by CGAcc, it will continue fetch data until all nodes are accessed. Simultaneously when a new node is found, which means the traversal order is confirmed, the node index will be sent back to CPU side.

ICPP 2018, August 13–16, 2018, Eugene, Oregon, USA

ACM ISBN 978-1-4503-5761-6/18/05…$15.00
https://doi.org/10.1145/3203217.3203235

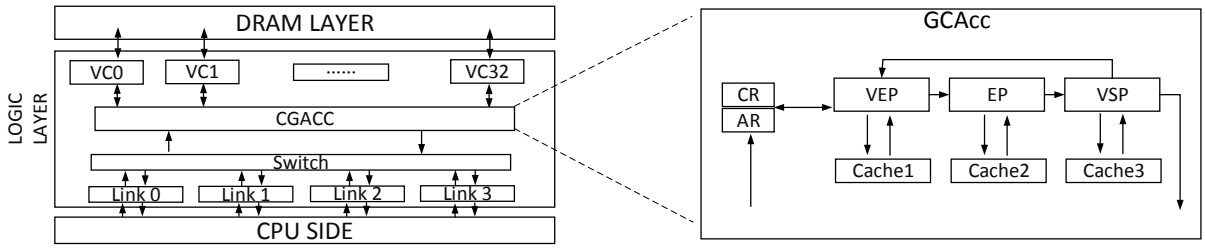

Figure 2: The architecture of HMC and CGAcc.

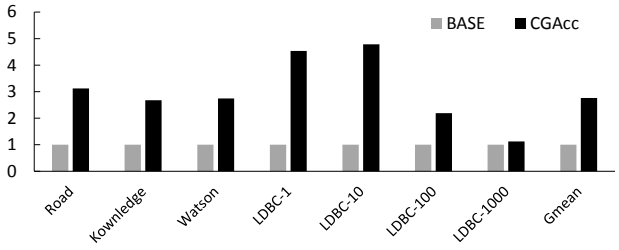

Figure 3: Comparison of performance on several synthetic and real-world workloads.

CGAcc modules overview CGAcc needs the support of several external modules as following shows.

1) A register (AR, Activation Register) which used as a trigger to wake CGAcc up. At the very beginning, CPU send an activation request which includes start node index to AR. CGAcc will access AR periodically before it is activated. Another register (CR, Continual Register) is used to store the current maximum start node index. This register is needed because in the vast majority cases, the graph can be treated as forest. Thus we set this register to start a new tree traversal when a tree traversal is completed.

2) On-chip Cache. In purpose of reducing transaction latency, we also deploy three caches (Call them C1, C2 and C3) on logic layer. These three caches are used as buffers for vertex, edge and visited array. On every memory access operation to different prefetcher, the corresponding cache will be access first. The data will be directly fetched if cache-hit happened, otherwise the related prefetcher will send a memory access to DRAM layer. In our current work, the caches obey to classical replacement policy. Actually it can be further optimized by knowing the nodes that will be accessed in the near future. This will be our future work.

3) Prefetcher group. Consisted of VEP (Vertex Prefetcher), EP (Edge Prefetcher) and VSP (Visited Prefetcher). VEP receives and uses new node index to access visited array and according to the result, fetch vertex data. Except accesses AR at the very beginning, VEP accesses CR when it is notified that the current tree is processed over. In other cases, VEP receives request that contains the new node index from VSP. When vertex data is fetched, VEP will send some requests to EP. EP is used to fetch edge data. It receives requests from VEP. After edge data (Extended by the current processing node) is fetched, EP will send request to VSP. VSP is used to fetch visited data. It receives request from EP and then it determines whether this node is new node by simply snooping if there exists a write access. Write access to visited array means a node that never be visited is now being visited, which also means that the node index should be sent to VEP for the following traversal.

3 EVALUATION

We implemented CGAcc in the cycle-accurate CasHMC simulator [4]. We used Intel PIN [5] to collect memory trace that traversal algorithm generates. This memory traces only contains the key part of the algorithm which excludes construction and initiation parts. A conventional memory system with two-level cache and stream prefetcher is used as baseline.

We use several graph benchmarks for evaluation. These benchmarks include synthetic and real-world graphs, provided by GraphBIG Dataset [7]. Synthetic graphs with flexible vertexes and edges (1k∼1000k in our experiment) are generated by LDBC-SNB data generator.

Figure 3 shows the evaluation results of CGAcc compared with baseline. The left three bars show the results of real-world benchmarks, and the middle four bars show the results of synthetic graphs. We can learn that for all benchmarks, the average speedup can reach 2.8X. Respectively, for real-world workloads, the average speedup is 2.84X while the speedup shows as 2.70X for synthetic workloads, which shows not much difference. We notice that, for workloads Road and LDBC-1000k which have similar vertex number, the speedups (2.67X and 1.12X) exist relatively large gap. This is because the acceleration is strongly related to the structure of graph. For instance, if the graph is dense enough, P2 may become the bottleneck for taking too much time fetching edge data.

4 ACKNOWLEDGEMENT

This work was supported by the NSFC under Grants 61472435 and 61572058, and YESS under Grant 20150090.

REFERENCES

[1] Robert Cooksey, Stephan Jourdan, and Dirk Grunwald. 2002. A stateless, content-directed data prefetching mechanism. In *ACM SIGPLAN Notices*, Vol. 37. ACM, 279–290.
[2] Eduardo F D'Azevedo, Mark R Fahey, and Richard T Mills. 2005. Vectorized sparse matrix multiply for compressed row storage format. In *International Conference on Computational Science*. Springer, 99–106.
[3] Babak Falsafi and Thomas F Wenisch. 2014. A primer on hardware prefetching. *Synthesis Lectures on Computer Architecture* 9, 1 (2014), 1–67.
[4] Dong-Ik Jeon and Ki-Seok Chung. 2017. Cashmc: A cycle-accurate simulator for hybrid memory cube. *IEEE Computer Architecture Letters* 16, 1 (2017), 10–13.
[5] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, Vol. 40. ACM, 190–200.
[6] 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 2010 ACM SIGMOD International Conference on Management of data*. ACM, 135–146.
[7] Lifeng Nai, Yinglong Xia, Ilie G Tanase, Hyesoon Kim, and Ching-Yung Lin. 2015. GraphBIG: understanding graph computing in the context of industrial solutions. In *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, 1–12.