DSAP: Data Structure-Aware Prefetching for Breadth First Search on GPU

Hui Guo†, Libo Huang†✉, Yashuai Lü‡, Qi Yu†, Sheng Ma† and Zhiying Wang†

†National University of Defense Technology, Changsha, Hunan, China
‡Space Engneering University, Beijing, China
{huiguo, libohuang, yashuailv, yuqi13, masheng, zywang} @nudt.edu.cn

ABSTRACT

Breadth First Search (BFS) is a key graph searching algorithm for many graph computing applications. However, memory divergence harms GPU's efficiency dramatically due to the irregular memory access pattern of BFS. Memory prefetching can fetch useful data into on-chip memory in advance, but current prefetchers cannot deal with data-dependent memory accesses. In this paper, we propose DSAP, a data structure-aware prefetcher on GPU that generates prefetching requests based on the data-dependent relationship between graph data structures and the memory access pattern of BFS. Also, we introduce an adaptive prefetching depth management to make DSAP more efficient according to the limited on-chip memory storage. We evaluate six datasets from three different kinds of applications and DSAP can achieve geometrical mean speedups of 1.28x, up to 1.48x, compared to the GPU without using prefetching.

Introduction

Breadth First Search (BFS) has been used in many graph computing algorithms, such as Page Rank, Single Source Shortest Path (SSSP) and other big data applications. As the input graph is increasingly larger, BFS becomes a performance bottleneck for most applications. The irregular memory access pattern of BFS is the primary reason that impacts its performance on GPU. When GPU processes these irregular memory accesses, the latency to fetch the entire data of a load instruction will be much longer than that of regular memory accesses. Even though GPU achieves latency hiding through massive parallelism, it still spends many stall cycles to wait for data.

To address the inefficiency of irregular memory accesses on GPU, prefetching is one of the promising technologies. Most of current prefetchers, such as stream prefetchers, stride prefetchers and GHO prefetchers, can effectively reduce memory latency for regular access patterns. MT-prefetching [1] proposed an inter-thread prefetching mechanism with hardware prefetchers training to reduce the negative effect of prefetching on GPU. APRES [2] combines warp scheduling and prefetching to improve cache efficiency. Both of the two recent proposed GPU prefetching mechanism are based on stride or stream prefetchers, which are not able to deal with indirect memory accesses of BFS.

CSR-format graph data structure is a kind of compressed graph data structures widely used to store large graphs. It uses two data arrays (vertex list and edge list) to describe the relationships of the vertex and edges in a graph. In this paper, we propose a Data Structure-Aware Prefetching (DSAP) that generates prefetching requests based on the data-dependent relationship of CSR-format graph data structures and the memory access pattern of BFS. DSAP can prefetch data-dependent data accurately with the explicit graph data structure information and improve the cache efficiency by the adaptive prefetching depth management. For six datasets from three different applications, DSAP achieves a geometric mean speedup of 1.28x, and up to 1.48x, compared to the GPU without prefetchers.

Data Structure-Aware Prefetching

A. Prefetching Operations

According to the traditional BFS algorithm, most of memory accesses are data-dependent. For example, accesses to vertex list is dependent on the vertex id fetched from the work list. However, the data dependent relationship and the memory access pattern are fixed and sequential, therefore prefetchers can easily copy its memory access pattern and generate prefetching requests according its data dependent relationships, with some explicit graph data structure information. Therefore, prefetchers need to monitor the memory accesses to decide which data to prefetch next. Tab. 1 shows how DSAP prefetches all the data of one iteration by monitoring memory accesses to graph data structures.

Table 1 Prefetching Actions Based on Monitored Memory Accesses

Monitored Memory Accesses	Prefetching Actions
Load vid = work_list[i]	Prefetching work_list[i+1]
Prefetched nvid = work_list[i+1]	Prefetching vertex_list [nvid] and vertex_list [nvid+1]
Prefetched eid = vertex_list[nvid] and neid = vertex_list[nvid+1]	Prefetching edge_list[eid:neid]
Prefetched edge_list[eid:neid]	Prefetching visited_list[edge_list[eid:neid]]

B. Adaptive Prefetching Depth

According to the traditional BFS algorithm, most of memory accesses are data-dependent. For example, accesses to vertex list is dependent on the vertex id fetched from the work list. However, the data dependent relationship and the memory access pattern are fixed and sequential, therefore prefetchers can easily copy its memory access pattern and generate prefetching requests according its data dependent relationships, with some explicit graph data structure information. Therefore, prefetchers need to monitor the memory accesses to decide which data to prefetch next. Tab. 1 shows how DSAP prefetches all the data of one iteration by monitoring memory accesses to graph data structures.

C. Hardware Design

For each cuda core, a DSAP prefetching unit is designed to sit aside L1 cache. Each prefetching unit has four main components (Fig. 1). A set of registers store the address boundaries of the graph data structures. A table records the runtime information of each warp. These runtime information includes the indices of graph data structures being accessed by the warp. DSAP uses this runtime information to calculate the addresses of data to be prefetched. A prefetching request queue stores the memory access requests generated by the prefetching unit. A control unit takes responsibilities for managing the adaptive prefetching depth and controlling prefetching.

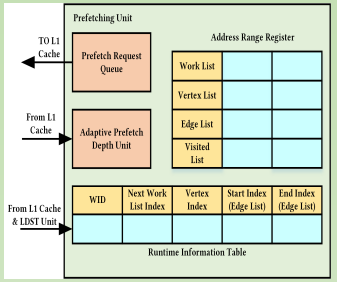

Figure 1 Hardware Components of DSAP Prefetching Unit

Evaluation

A. Methodology

We implement the GPU with the prefetching unit on GPGPUSim simulator [4]. And the parameters of the simulator are based on GTX-480, listed in Tab.2. The datasets we test are from the SNAP datasets [3] and their features are listed in Tab. 3.

Table 2 The Parameters of GPGPUSim Simulator

Parameter	Value
Number of SMs	15
Threads per SM	1536
Threads per Warp	32
Warp Scheduling Policy	GTO
L1 Cache Size	48KB

Table 3 The Features of Tested Datasets

Name of Dataset	Nodes	Edges
roadNet-CA	1,965,206	27,766,607
USA-Road	1,070,376	2,712,798
roadNet-TX	1,379,917	1,921,660
Oregon-2	11,461	32,731
Cit-HepPh	34,546	421,578
roadNet-PA	188,092	1,541,898

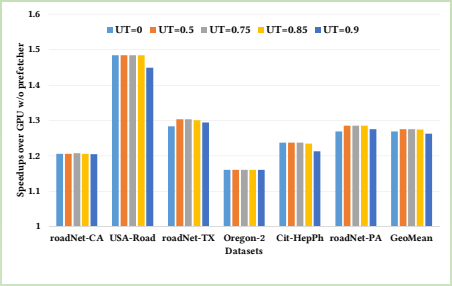

Figure 2 Speedups of DSAP with different UTs for six datasets

B. Adaptive Prefetching Depth

Fig. 2 shows the speedups of BFS running on the GPU with the prefetchers using different utilization thresholds. "UT" means Utilization Threshold, which is used by the adaptive prefetching depth management. When UT equals to 0, the adaptive prefetching depth management is inactive. In general, we test six datasets from three different applications, which are road networks, autonomous systems graphs and citation networks and DSAP achieves a geometric mean speedup of 1.28x, compared to the performance of the GPU without prefetchers. For the USA-Road dataset, DSAP achieves the highest speedup of 1.48x, while for the Oregon-2 dataset, it only has a speedup of 1.16x. The size of the graph is a key factor. The size of Oregon-2 is small enough to be cached in L2 cache, therefore DSAP only benefits the cycles of transferring data from L2 to L1. When UT equals to 0.75, DSAP achieves the best performance, this demonstrates the adaptive prefetching depth management can make a good balance between prefetching data and on-chip memory storage.

REFERENCES

1. Lee, J., Lakshminarayana, N. B., Kim, H., and Vuduc, R. 2010. Many-thread aware prefetching mechanisms for GPGPU applications. In Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on (pp. 213-224). IEEE.
2. Oh, Y., Kim, K., Yoon, M. K., Park, J. H., Park, Y., Ro, W. W., and Annavaram, M. 2016. APRES: improving cache efficiency by exploiting load characteristics on GPUs. ACM SIGARCH Computer Architecture News, 44(3), 191-203.
3. https://snap.stanford.edu/data/index.html
4. Bakhoda, A., Yuan, G. L., Fung, W. W., Wong, H., and Aamodt, T. M. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on (pp. 163-174). IEEE.

Acknowledgements

This work was supported by the NSF of China under Grants 61433019, 61472435, 61572058, 61672526, 61202129, U14352217, YESS under Grant 20150090 and Research Project of NUDT ZK17-03-06.