GPU 에서 애플리케이션의 데이터 액세스 패턴 프로파일링

테오도라 아두푸, 김윤희*
컴퓨터과학과, 숙명여자대학교

theoadufu@sookmyung.ac.kr, *yulan@sookmyung.ac.kr

Profiling Data Access Patterns of Applications on GPUs

Theodora Adufu, Yoonhee Kim*
Department of Computer Science, Sookmyung Women's University

Abstract

Memory management is a significant aspect of executing applications on GPUs. With the advancements in GPU architecture, issues such as data reuse, cache line eviction and data residency are to be considered for optimal performance. Frequency of data access from global memory has significant impacts on the performance of the application with increased latencies when accesses result in cache misses. Through static profiling, we identify the access patterns to the global memory and investigate the relationship between frequent access patterns and data residency in the cache. From our investigations, we observed that each application frequently accesses a data region in memory though the range of addresses accessed differ. *Index Terms*—Static Profiling, Frequently Accessed Data, Data Residency.

Ⅰ. INTRODUCTION

Graphics Processing Units (GPU) provide high computational capacity for compute intensive applications such as High Performance Computing (HPC) applications. However, there remains a bottleneck in performance mostly as a result of misaligned memory accesses leading to high miss rates. To alleviate this phenomenon, researchers [1][2][3] have proposed the use of different approaches which including the use of FIFO buffers [4] to reorder memory requests and as a result shorten the reuse distance of memory requests before they are sent to L1 caches.

NVIDIA, with its Ampere architecture for instance, offers a new feature that allows the user to leverage data persistence in a defined portion of the L2 cache for applications with high frequent accesses [5]. This is an effort to reduce early eviction of data thus ensuring that data that is frequently accessed is available during the execution lifetime of the application hence lowering access latencies.

This paper proposes the use of static profiling of PTX code to determine the access frequencies of data read from the global memory. From this, regions that are frequently accessed can be identified and applications can be classified into either streaming normal or persistent categories based on the access frequencies to the data regions. The range of frequently accessed memory addresses can then be marked for persistent data storage. Through this study,

- We determine the data access frequencies of applications through static profiling and create data access profiles for the applications
- We classify the applications into three groups; persistent, normal and streaming based on a frequency score

The proposed approach provides a basic application classification for determining data residency configurations for different applications on modern GPU architectures.

The rest of the paper is organized as follows: in Section 2, we briefly describe the research works related to this study. In Section 3, we give details of the proposed static profiling approach for determining the data access profiles as well as the application classification method. In Section 4, we present the results of our study and conclude the paper in Section 5.

Ⅱ. RELATED WORKS

The designs of modern GPU architectures reveal an attempt to maximize memory bandwidth by using as much fast memory and as little slow-access memory as possible hence improving over-all performance. Prior research works [6][7][8][9] have attempted to identify access patterns and analyze data reusability between thread blocks to maximize the gains from data locality among threads. Also according to Walden et al.[2], the data layout of applications influence the effective utilization of memory bandwidth in GPU architectures. In order to maximize the benefits of new features introduced in modern GPU architectures such as the L2 cache residency control feature, it is imperative to quantitatively determine the amount of frequent accesses by the application and identify the access patterns to the global memory. Degioanni's StAMP[10], propose a memory access profile which can be used by off-line scheduling strategies to minimize interference overhead. However, they did not consider the frequency of access to data regions.

In our research, creating a data access profile serves as a basis for classifying an application. From the data access profiles, data regions with continuous access frequencies can be identified and explored to influence data residency decisions.

Ⅲ. DATA ACCESS PROFILES BASED ON STATIC PROFILING

In this section, we describe our data access profiling approach. We begin the process by assembling a PTX code from the application's executable and use a modified PTX parser obtained from [11] to obtain information for the data access profile. The data access information is obtained in a two-step process.

The first step requires the building of a syntax tree [7]. The syntax tree derives the thread-to-memory-addresses accessed relationship in terms of thread ID, block ID and other kernel parameters. This information can be used to capture inter-thread, inter-warp, inter-TB locality within the same kernel as well as across multiple kernels.

Secondly, from the syntax tree, the addresses accessed from the global memory by threads and the frequency of accesses is obtained with the **ld.global** command.

A. Data Access Granularity

Blocks are divided into warps of 32 threads with every thread in the warp executing the same instruction in lock-step manner but on different data. When a warp executes an instruction that accesses memory, the requests are processed together for all the threads within the warp. Thus, we extract the access frequencies at the warp granularity. With our static profiling approach, we consider each entry and exit of threads to a given data region in the global memory as an access order and do not consider multiple contiguous loads from the same data region separately.

B. Data Access Profile

Using the information obtained, we create a data access profile for each application. The data access profile is expressed both graphically and in tabular form. The data access graph shows only the application's access patterns and frequencies. However, the data access profile table contains additional information derived from further analysis. This includes the Frequently Accessed Address Range **(FAAR)**, the Sum of regions within FAAR, the total memory regions, the Frequency Score **(FS)** and the class of the application.

The Frequently Accessed Address Range **(FAAR)**, is the memory address region frequently accessed by the application during the application's life-cycle, in bytes. From the data access graph, this can be seen as the dense parts of the graph. The number of regions within this repeatedly accessed data region accessed throughout the execution of the application is known as the **(Sum of regions in FAAR)**. This metric is particularly useful in determining the class of the application as well as the **Total Memory Regions** accessed by the application.

C. Application Classification Approach

For quantitative analysis and application classification, we calculate a Frequency Score (FS), which is the ratio of all memory regions accessed in The Frequently Accessed Address Range (FAAR) to the Total Memory Regions accessed by the application as shown in Equation 1.

$$\text{Frequency score, FS} = \frac{\sum \text{Regions in FAAR}}{\text{Total Memory Regions}} \qquad (1)$$

Based on the Frequency Score, applications can be classified into one of three classes: streaming(S), normal (N) and persistent (P). We define these three classes according to NVIDIA's caching policies [12] and apportion an FS score range to each class. For a score within the range, $0<FS<0.33$, the application is classified as streaming. For a score within the range $0.33<FS<0.66$, the application is classified as normal. When data is accessed frequently giving an FS score within the range $0.66<FS<1$, the application is classified as persistent. Our classification is highly dependent on the frequency of data access throughout the application's execution life-cycle which serves as a reliable measure.

IV. EXPERIMENTS AND RESULTS

We statically profile four applications: LSTM, HISTOGRAM, GEMM and BICG from the Tango Benchmarks [13], NVIDIA Cuda-Samples [14] and Polybench benchmarks [15] respectively on the NVIDIA A30 (Ampere architecture) using 1 GPU.

TABLE I

APPLICATION GRID-BLOCK DIMENSIONS

APPLICATION	GRID X	GRID Y	THREAD X	THREAD Y
LSTM	1	1	100	1
HISTOGRAM	128	64	64	1
GEMM	2	8	32	8
BICG	256	16	16	8

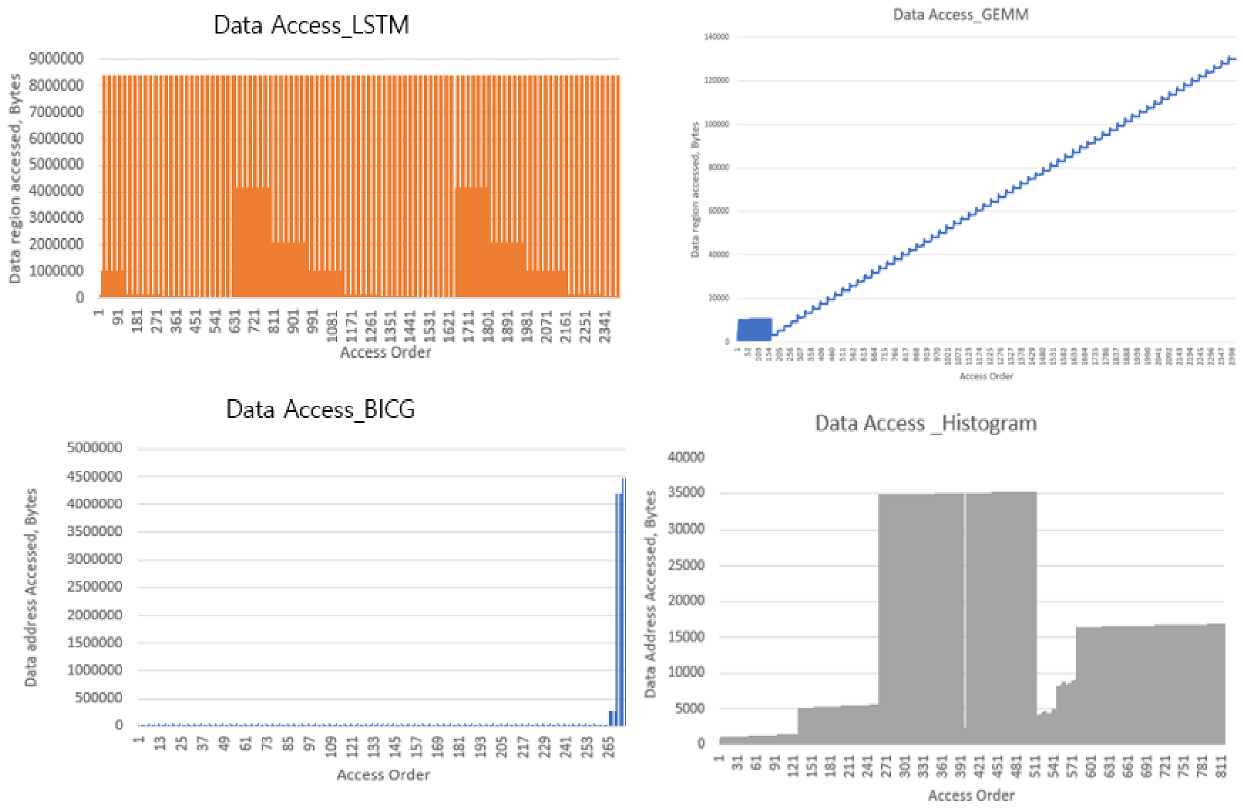

Fig. 1. Data access graphs for selected applications

TABLE II
DATA ACCESS PROFILE TABLE

APPLICATION	FAAR, B	TOTAL MEMORY REGIONS	SUM OF REGIONS IN FAAR	FS	CLASS
LSTM	4197496	2406	2005	0.83	P
HISTOGRAM	16892	816	688	0.84	P
GEMM	10364	2416	292	0.12	S
BICG	51324	276	264	0.96	P

We compiled each application with CUDA version 12.0 before generating the PTX code. Table I shows the grid/block dimensions of the workloads used during the static profiling analysis.

A. Observation 1: Frequently Accessed Address Region

From Figure 1, we observe that, all applications frequently access data within a given range. The range however varies for each application. LSTM application for instance has a uniformly repeated access pattern to data regions up to 4197496 B (4MB) though it accesses data over a 7.6 MB range.

GEMM on the other hand accesses a range of 10364 B (10KB) repeatedly at the beginning of the execution and later streams data from different memory locations up to 127 KB.

B. Observation 2: Frequency Score (FS) and Classification of applications

The frequency score for each application is calculated relative to the range of data regions accessed by the application. From Table 2, we observe that BICG application has the highest FS of 0.96 showing that most of the data it accesses is within the given range identified as the frequently accessed region. The size of the frequently accessed memory region, 50KB, is however very small compared to that of LSTM, 4MB. Though the size of frequently accessed regions for HISTOGRAM and GEMM is similar, HISTOGRAM has a higher FS of 0.84 which corresponds to the access patterns depicted in the graph. GEMM on the other hand can be classified as a streaming application as shown in the graph and from the FS score of 0.12.

V. CONCLUSION AND FUTURE WORK

This paper uses static profiling analysis to identify for selected applications, the access patterns to global memory when executed on NVIDIA's A30 GPU. From our investigations, we observed that each application accesses a given memory region repeatedly. We classify the applications into three groups based on the frequency of access throughout the life-cycle of the application.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No. 2021R1A2C1003379).

참 고 문 헌

[1] Duong, Nam, et al. "Improving Cache Management Policies Using Dynamic Reuse Distances." 45th Annual IEEE/ACM International Symposium on Microarchitecture, Pages 389-400, 2012

[2] A. Walden, M. Zubair, C. P. Stone and E. J. Nielsen, "Memory Optimizations for Sparse Linear Algebra on GPU Hardware," 2021 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC), 2021, pp. 25-32, doi: 10.1109/MCHPC54807.2021.00010.

[3] D. Tripathy, A. Abdolrashidi, L. N. Bhuyan, L. Zhou, and D. Wong, "Paver: Locality graph-based thread block scheduling for gpus," ACM Transactions on Architecture and Code Optimization (TACO), vol. 18, no. 3, pp. 1- 26, 2021.

[4] Fang Juan, Zelin Wei, and Huijing Yang. 2021. "Locality-Based Cache Management and Warp Scheduling for Reducing Cache Contention in GPU" Micromachines 12, no. 10: 1262. https://doi.org/10.3390/mi12101262

[5] NVIDIA A100 datasheet, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf,

[6] Danqi Wang; Chai Kiat Yeo, "Exploring Locality of Reference in P2P VoD Systems," Multimedia, IEEE Transactions on, vol.14, no.4, pp.1309, 1323, Aug.2012

[7] D. Tripathy, A. Abdolrashidi, Q. Fan, D. Wong, and M. Satpathy, "Localityguru: A ptx analyzer for extracting thread block-level locality in gpgpus," in 2021 IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 1- 8, IEEE, 2021.

[8] Di Carlo, S.; Prinetto, P.; Savino, A., "Software-Based Self-Test of SetAssociativeCache Memories," Computers, IEEE Transactions on , vol.60, no.7, pp.1030,1044, July 2011 doi:10.1109/TC.2010.166

[9] Fensch, C.; Barrow-Williams, N.; Mullins, R.D.; Moore, S., "Designing a PhysicalLocality Aware Coherence Protocol for Chip Multiprocessors," Computers, IEEE Transactions on , vol.62, no.5, pp.914,928, May 2013

[10] Théo Degioanni, Isabelle Puaut. StAMP: Static Analysis of Memory access Profiles for real-time tasks. WCET 2022 - 20th International Workshop on Worst-Case Execution Time Analysis, Jul 2022, Modena, Italy. □10.4230/OASIcs.WCET.2022.1□. □hal-03723457□

[11] PTX Parser,https://github.com/JIeunAmy/coalescing_

graph_with_PTX

[12] Kernel Profiling Guide, https://docs.nvidia.com/nsight-compute/ProfilingGuide/

[13] Karki, Aajna Keshava, Chethan Shivakumar, Spoorthi Skow, Joshua Hegde, Goutam Jeon, Hyeran. (2019). Tango: A Deep Neural Network Benchmark Suite for Various Accelerators.137-138. 10.1109/ISPASS.2019.00021.

[14] CUDA SAMPLES, https://github.com/NVIDIA/cuda-samples/

[15] Louis-Noël Pouchet. 2012. Polybench: The polyhedral benchmark suite. http://www.cs.ucla.edu/pouchet/software/polybench.

[16] Adufu, T. and Kim Y. (2022). A Performance Benchmark of Cached Data Access Patterns on GPUs. KNOM Review '22-02 Vol.25 No.02, pg 30-39. https://doi.org/10.22670/knom.2022.25.2.

[17] NSIGHT COMPUTE, https://developer.nvidia.com/

nsight-compute