IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 June 2023, accepted 11 August 2023, date of publication 21 August 2023, date of current version 8 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3307315

RESEARCH ARTICLE

Analyzing Data Locality on GPU Caches Using Static Profiling of Workloads

JIEUN KIM[1], HYEONSANG EOM[1], AND YOONHEE KIM[2]
[1]Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Republic of Korea
[2]Department of Computer Science, Sookmyung Women's University, Seoul 04310, Republic of Korea

Corresponding author: Yoonhee Kim (yulan@sookmyung.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government, Ministry of Science and ICT(MSIT) (No. NRF-2021R1A2C1003379).

ABSTRACT The diversity of workloads drives studies to use GPU more effectively to overcome the limited memory of GPUs. Precisely, it is essential to understand and utilize data locality of workloads to utilize the memory and cache efficiently, which is relatively smaller than CPU's. It is important to understand GPU memory hierarchy to efficiently use with multi-thread environment. Although there have been previous approaches to analyzing data locality on GPUs, these approaches focused on global memory and L2 cache levels with profiling at thread block levels. Data locality study in warp level in GPU has not been studied much. Especially, the concept of coalescing has been defined but the method of measuring the `degree of coalescing` has not been discussed. Our study focused on analyzing data locality in L1 cache levels, which is the smallest but fastest in cache level to analyze the impact of data locality. To achieve this analysis, our study profiles data locality in warp level, which is smallest segment in GPU thread groups. This paper introduces a novel perspective by introducing a quantitative measure for coalescing alongside static profiling of data locality. Furthermore, it offers a means of refining locality estimates by scrutinizing access patterns of L1 cache. To substantiate our approach, our study validates the estimated data locality against a range of real-world GPU benchmarks, including Rodina and Polybench. Through empirical experimentation, our results reveal a substantial correlation between the metrics of data locality and cache utilization, affirming the efficacy of our proposed method.

INDEX TERMS Data locality, GPU cache, GPU profiling, GPGPU workload analysis, PTX code.

I. INTRODUCTION

Graphics processing units (GPUs) are widely used as GPGPUs (general-purpose computing on graphics processing units) in a variety of fields, such as high-performance computing, machine learning, and big data analysis. With the recent increase in types in workloads executed by GPUs and the quantity of data utilized in workloads, memory access efficiency of workloads executed using GPUs has become a crucial topic that merits extensive research. Efficient utilization of GPUs during workload execution requires a thorough understanding of their parallel computing characteristics and memory hierarchy.

The associate editor coordinating the review of this manuscript and approving it for publication was Nikhil Padhi.

In a GPU, multiple threads are grouped into a single execution unit, and they execute each instruction simultaneously. As a result, data locality occurs for each memory hierarchy whenever a group of threads executes memory access instructions simultaneously. This data locality can be analyzed to understand the data access characteristics of the workload and predict access efficiency in the actual memory hierarchy. Data locality occurring in GPUs can be classified in terms of the execution method. While executing a workload, an NVIDIA GPU specifies a group of threads to be used. A hierarchy exists within the group of threads-the largest unit is a grid consisting of thread blocks, and each thread block, in turn, consists of threads.

Although the sizes of the grid and the thread can be specified by the user while executing a workload, each thread block

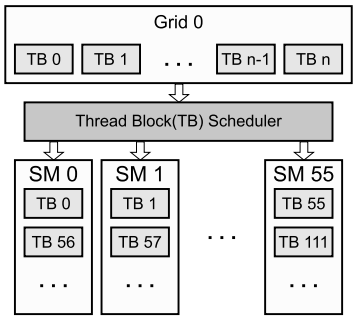


FIGURE 1. Example of scheduling in SM of thread block.

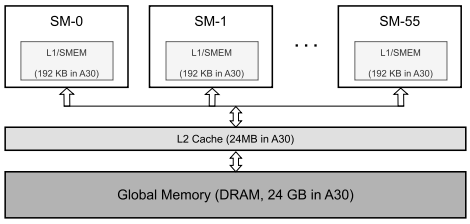


FIGURE 2. The memory hierarchy of NVIDIA A30.

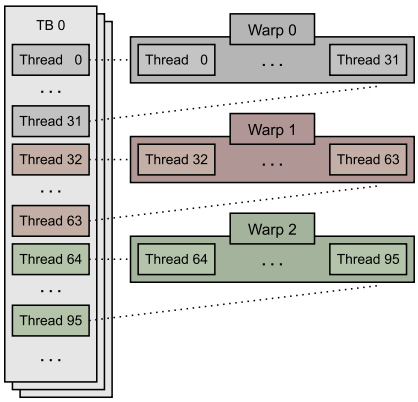


FIGURE 3. Warp scheduling in thread block.

comprises 32 threads partitioned into warps during actual execution. All threads in a warp execute each instruction simultaneously. This creates various types of data locality during simultaneous execution of each instruction by multiple threads in thread groups. Firstly, data locality can occur between thread blocks within the grid. Figure 1 depicts various thread blocks scheduled on streaming multiprocessors (SMs). As depicted in Figure 2, SMs share the L2 cache of the GPU. Therefore, significant overlap of data access between SMs increases data locality between them, increasing L2 cache access efficiency. Secondly, as L1 cache exists in each SMs, data locality can occur in SM level. Each SM is composed of thread blocks as depicted in Figure 1, which means at minimum, data locality within thread block effects data locality in L1 cache.

As illustrated in Figure 3, several thread blocks exist within the SM, and each thread block is further subdivided into warps. Data locality within the SM is influenced by that between thread blocks and within warps. In order to utilize a GPU architecture effectively for execution by grouping threads into certain units, global memory accesses is coalesced. When a warp executes a global memory access instruction, the efficiency of global memory access is determined by the arrangement of the memory addresses accessed by the threads within it. One of the criteria for classifying this arrangement of memory addresses accessed involves verifying whether the memory addresses accessed by the threads within the warp are contiguous. When the data type size is 4 bytes, the optimal access byte size accessed by each warp is 128 bytes. One sector consists of 32 bytes, with fewer access sectors corresponding to more common accesses occurring within the cache line. On the other hand, memory accesses with higher numbers of access sectors and bytes are more distributed and inefficient [9].

Nvidia guide explains the concept of coalesced data access in GPU [8]. As depicted in Figure 4, 4(a) reveals that threads within a warp enable consecutive accesses. In this case, in order for a warp to complete the global memory access, all threads within it complete the execution by accessing 4 sectors simultaneously. On the other hand, in the case of 4(b), consecutive accesses are performed, but the range does not fit within the 32-byte sector unit. As a result, 5 sectors must be accessed, which is a drawback. Further, in the case of 4(c), the threads of the warp access memory by skipping memory locations over regular intervals. This incurs additional overhead for accessing multiple sectors. Besides these cases, cases involving overhead owing to threads not accessing memory in units of 128 bytes are referred to as non-coalescing access. There has been previous research on this to analyze the effect of this characteristic on GPU workloads.

In Localityguru [13], PTX code-based static profiling was performed. Data locality between thread blocks was analyzed by the memory address which threads accessed, using this as the parameter to schedule the thread blocks with high locality. In addition, several studies have been conducted to analyze data locality [1], [2], [3], [4], [14] and the results have been applied to GPU scheduling [12], [13]. In particular, LocalityGuru introduced a method for determining data locality between mutual thread blocks using static profiling based on PTX code. This method can identify data access addresses that overlap between the thread blocks. Further, the authors proposed a thread block locality graph to identify thread block combinations with high data locality.

Although the previous papers mentioned above analyzed data locality,the relationship between memory and cache, there were limitations in the analysis at the warp level and the L1 cache analysis on the GPU. Data locality analysis at the thread block level is limited in terms of associating data locality with multiple level of cache usability. For instance, only threads within the same SM can access the L1 cache. However, examining data locality at the thread block level does not fully reflect the characteristics of GPU threads

fetching data from global memory. To achieve this L1 cache level analysis, this paper narrow the analysis scope to warp level. In GPU, the same instruction is executed in warp units. In the case of global memory access, the distribution of data locations accessed by each thread within the warp affects the performance significantly. This can be expressed using data coalescing, as well as the contiguity of data accessed by the threads within the warp and the density of their location affect performance significantly. Therefore, this paper aims to predict the actual cache usage pattern by analyzing data locality at the warp level.

To this end, our study analyzes the memory access patterns of workloads using static profiling based on PTX code. This paper also quantify the degree of data locality and coalescing and ascertain the correlation between the quantified values and the dynamic profile result. NVIDIA A30 GPU is used in the experiment to evaluate the relationship between the `degree of coalescing`, GPU L1 cache, and the number of sectors/requests required for data access instructions for the rodinia [10] and polybench [11] workloads using PTX code-based static profiling, as presented in [13]. Our study observes that 6 out of 6 workloads with coalescing scale values exceeding 70% exhibit L1 cache hit rates exceeding 60%. In addition, the scale of coalescing and the cache hit rate are observed to be correlated. hlIn addition, it was confirmed that the predicted sector access result using the value obtained through profiling has a high correlation with the actual sector per request. Based on these result, static profiling reveals that data locality of the threads in a warp affects the L1 cache hit rate and sectors/requests.

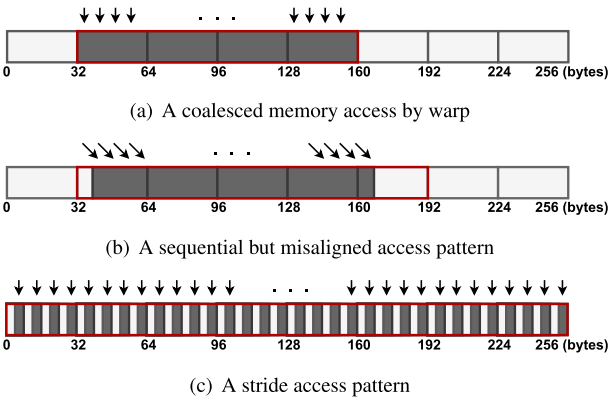
The contributions of our work are as follows:

- By analyzing data locality at the warp level, the analysis of data access patterns at a L1 cache level is objective.
- The estimation of data locality with degree of coalescing is compared to related metrics including L1 cache usage from Nvidia Nsight Compute profiling results.
- The graph method of visualizing data locality including the degree of coalescing in a warp level has been proposed. It helps to understand the coalescing characteristics of workloads.

The rest of this paper is organized as follows: Section II describes the related work. Section III presents the implementation of our data locality profiling. Section IV shows the experimental results. Section V concludes this paper.

II. RELATED WORK

Figure 1 illustrates the grouping of threads, i.e., the GPU execution units, into thread blocks and their scheduling on the SM. Thread blocks are bundled into grid units. The threads in each thread block comprise a warp consisting of 32 threads, and they execute each instruction simultaneously. Due to this GPU characteristic, data locality occurs among threads, warps, and thread blocks. Data locality can be subclassified as intra-locality and inter-locality. Intra-locality refers to locality within a thread block or SM. If multiple threads within a thread block access the same memory address, the thread block exhibits high intra-locality. On the other hand, inter-locality occurs when threads from different thread blocks or SMs access the same memory addresses. High inter-locality affects the L2 cache level that can be accessed by all threads.



(a) A coalesced memory access by warp

(b) A sequential but misaligned access pattern

(c) A stride access pattern

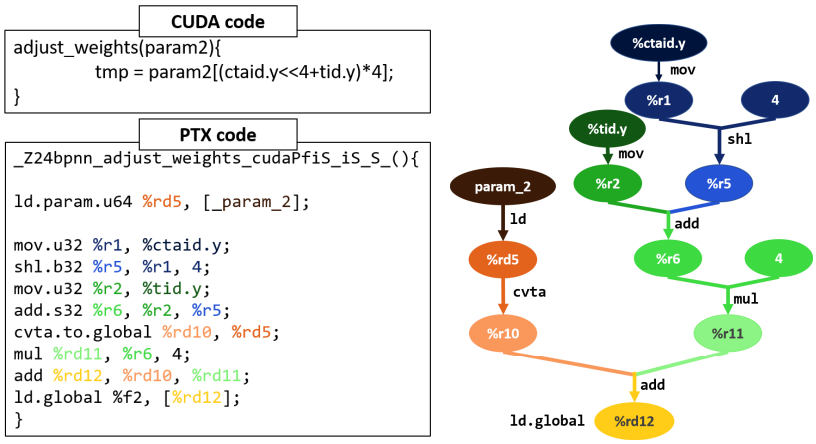
FIGURE 4. Warp memory access type.

A. STATIC PROFILING ON DATA LOCALITY IN GPU

Studies have been conducted to identify the characteristics of workloads by analyzing the locality in them. References [1], [2], [3], [4], [12], and [13]. LocalityGuru [13] analyzed PTX code to define the degree of memory access overlap between thread blocks as locality and analyzed degrees of locality between thread blocks. It is evident from Figure 1 that several thread blocks are assigned to one SM in the GPU. In this case, the thread blocks share the L1 cache within the SM. As a result, the locality between the threads affects the L1 cache. Considering these characteristics, the aforementioned paper analyzed the inter-locality between thread blocks, which identifies the degree of data reuse between the thread blocks. PAVER [12] extended the aforementioned results using the locality information during scheduling. Applying the techniques on scheduling the threads, [12] reduced L2 accesses while increasing the benchmarks' performance.

B. DATA ACCESS PATTERN ANALYSIS

There were previous studies focusing on profiling data locality and cache in GPU [19], [20]. Tang et al. [19] propose a cache miss analysis model for GPU for the first time. The method is focused on analyzing thread level stack distance and cache contention. The experimental results illustrate that their method can give the guidance in optimizing cache locality for the GPU programs. Nugteren et al. [20] extends stack distance or reuse distance theory on sequential processors to parallel processor,GPU. Nugteren [20] considers multi-thread aspects of GPU such as the hierarchy of threads, warp divergence, cache associativity, and so on. This paper showed L2 cache usage improvement when using their methods. Both of the studies focused on GPU cache, but experimented on gpgpu-simulation, which is virtual GPU. Additionally, Hong



(a) Backprop cuda code and ptx code (b) Backprop syntax

FIGURE 5. Backprop workload's code and syntax tree.

et al. [18] apply machine learning technique to analyze model data access patterns. It is the first work that employs deep learning techniques to flow visualization challenges. Also, this approach employs CPUs and GPUs together for particle tracing tasks. In addition to profiling only for GPUs, there are more approaches to processing data access pattern analysis with machine learning-based technology [15], [16], [17].

III. THE DESIGN OF DATA LOCALITY ANALYSIS AND IT'S IMPLEMENTATION

A. DATA LOCALITY ANALYSIS THROUGH PTX CODE

This paper performs static profiling based on PTX code-based locality analysis proposed in LocalityGuru [13]. In [13], the inter-locality between thread blocks during thread block scheduling was analyzed with a focus on locality. However, this paper analyzes the `degree of coalescing` between the threads within a warp based on the PTX code by reducing the scope of locality analysis to warp-level coalescing.

B. LOAD GLOBAL-BASED SYNTAX TREE GENERATION METHOD

A locality graph is generated based on a syntax tree, which is created while interpreting the PTX code. The PTX code consists of a list of instructions, an example is depicted in Figure 5. The PTX code may be expressed as a tree resembling the tree diagram depicted in Figure 5(b).

As depicted in Figure 5(a), PTX instructions are constructed as follows:

$$operation\ dst,\ src1,\ src2(,\ scr3)$$

A PTX instruction consists of an operation to be executed, the destination where the execution result is stored, and a source that executes the instruction. There can be maximum 3 sources, depending on the type of operation. Based on this PTX code, a syntax tree is created by choosing 1 instruction from the PTX code and recursively following it to detect the source used to create the destination of the instruction. In [13], a syntax tree was created based on instructions with the operation `ld.global` to determine the locality of memory access. Since `ld.global` commands are instructions that access memory and read data from it, the load global (`ld.global`) operation is the starting point of tracing and syntax tree was constructed by tracing the PTX code and analyzing the creation of the memory access address of this instruction. This is repeated until the kernel's input parameter, input matrix, or fixed parameters, such as thread ID and thread block ID, are encountered. Then, the information about the memory address accessed by the thread was derived using the traced syntax tree.

Figure 5(a) depicts some of the PTX commands of the backprop workload taken from the rodinia benchmark [10]. Among these, `rd12`, which is a register that accesses global memory, is adopted as the reference before initiating tracing. If `rd12` is found, the "`add rd12, rd10, rd11`" command, whose destination is this register, is searched for. Subsequently, the same operation is performed on the source registers of this instruction, i.e., `rd10` and `rd11`. This process is performed for all load global commands to trace all load global commands in the PTX code to generate the syntax tree.

C. WARP LEVEL DATA LOCALITY ANALYSIS METHOD

As Figure 3 shows that warp is consisted of a series maximum 32 threads, analyzing the locality at the warp level involves analyzing the locality of all 32 threads in the warp. This requires clear definition of their areas. The area that determines the `degree of coalescing` within the warp is defined as the coalescing range, which lies within 128 bytes from the starting position of the sector accessed by the first thread of the warp. Here, a sector refers to a group of 32 bytes.

As the access range depends on the data type size, coalescing is analyzed in this study by assuming a constant data type size of 4 bytes. As explained in Figure 6, the starting point

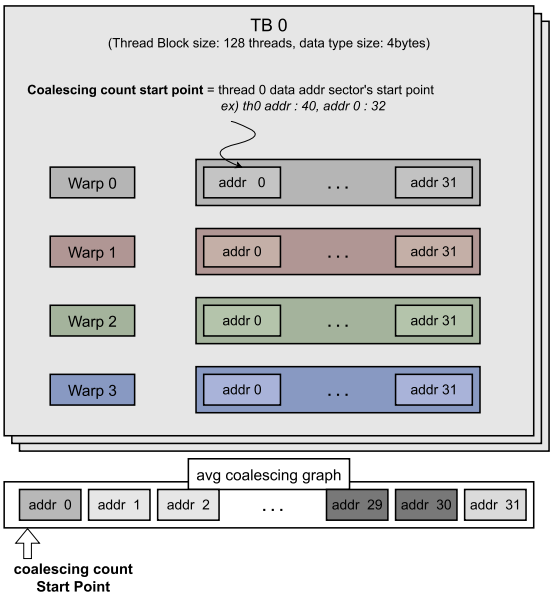


FIGURE 6. Process of coalescing graph generation.

of the coalescing graph does not correspond to the address accessed by the first thread of the warp. It corresponds to the starting point of the sector that includes that address. The starting point of the coalescing graph is defined in this way because the cache of the GPU is sectored; hence, when data access is required, data is read from the global memory in sector units. In other words, the entire sector including the data is read, instead of reading just the data.

Therefore, the starting point of the sector of the accessed data becomes the starting point for generating the coalescing graph. The terminology to define the `degree of coalescing` is as follows:

- `first_address`: global address of warp's first thread
- `coalescing_count_start_point`

$$= \left\lfloor \frac{first\ address}{32} \right\rfloor \times 32$$

- `cache_line_count`: total thread count / 32
- `total_coalescing`: total coalescing thread count within all warps
- `global_inst_count`: count of global instructions in PTX code

The starting point of the sector indicates the value obtained by multiplying the quotient obtained when dividing the access address by 32, by 32. With this starting point as the reference, a coalescing graph is created by enumerating the number of threads in the warp that access memory addresses within 128 bytes of the starting point.

$$degree\_of\_coalescing = \frac{total\_coalescing}{(cache\_line\_count \times global\_inst\_count)} \quad (1)$$

TABLE 1. The specification of NVIDIA A30.

NVIDIA A30 spec	
L1 cache size	192KB
L2 cache size	24MB
Memory size	24GB
Compute capability	8.0

Subsequently, the `degree of coalescing` is calculated by dividing the total number of threads by the number of instructions accessing the global memory and line count. Then the above information is expressed as a graph. The coalescing graph represents how the threads access data in 32 memory addresses with reference to the starting point of the sector accessed by the first thread of a warp.

The `degree of coalescing` and the number of sectors accessed can be predicted based on our graph. A sector is a unit of memory that is 32 bytes in size and is arranged in a specific way within a cache line or device memory. In the case of an L1 or L2 cache line, 4 of these sectors are grouped together, resulting in a total size of 128 bytes [9]. Based on this fact, the number of sectors accessed by the warp can be predicted, considering coalescing graph as representation of all threads' data accessing pattern.

IV. EVALUATION

A. EXPERIMENTAL SETUP

NVIDIA's A30 GPU is used for the experiments. Its memory hierarchy is configured as illustrated in Table 1. The size of the L1 cache in the A30 GPU is 192 KB, and each SM contains 1 L1 cache. Hence, the L1 cache is only accessed by threads scheduled on the same SM. The size of the L2 cache in the A30 GPU is 24 MB, and all threads in the workload can access the L2 cache. The size of data accessed at any time depends on the cuda computation capability. Version 8.0 is used, and data are grouped into 32-byte segments and accessed in the global memory. If the data type size is 4 bytes, the ideal number of sector accesses for a global memory load request is 4 [8]. If the same data tends to be accessed repeatedly, the number of accesses may be reduced. In the worst case, all threads in a warp may access data in other sectors, leading to 32 sectors being accessed for a single global memory request [8].

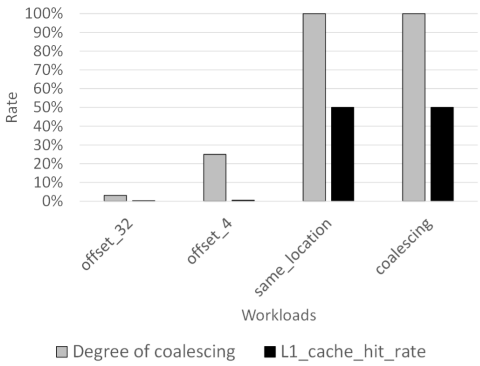
The first 4 workloads in Table 2 (stride_32, stride_4, same_location, and coalescing) are baseline workloads created to be compared to benchmark workloads selected from Rodinia [10] and Polybench [11]. These baseline workloads aim to illustrate the tendency of locality and coalescing. The other 9 workloads selected from Rodinia [10] and Polybench [11] are chosen to provide diversity in experiments conducted under various conditions. They include workloads with multiple kernels such as BFS, backprop, and b+tree, each having different grids and thread features (refer to Table 2).

TABLE 2. The grid and thread block size according to workloads.

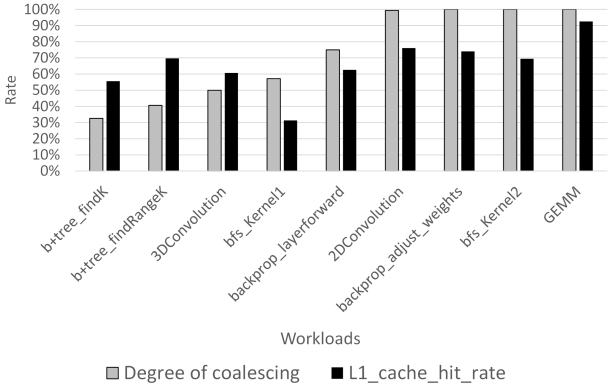
Workload	Grid_x	Grid_y	Thread_x	Thread_y
stride_32	32	1	64	1
stride_4	32	1	64	1
same_location	32	1	64	1
coalescing	32	1	64	1
b+tree_findK [10]	60	1	256	1
b+tree_findRangeK [10]	60	1	256	1
3DConvolution [11]	4	16	16	4
bfs_kernel1 [10]	128	1	512	1
backprop_forward [10]	1	64	16	16
2DConvolution [11]	2	8	32	8
backprop_adjust_weights [10]	1	64	16	16
bfs_kernel2 [10]	128	1	512	1
GEMM [11]	2	8	32	8

TABLE 3. Degree of coalescing, sector expectation and sector per request.

Workload	Degree of coalescing	Sector access within coalescing range	Estimated sector access	L1 sectors per request
stride_32	3.13%	1	32	32
stride_4	25.00%	4	16	16
same_location	100.00%	1	1	1
coalescing	100.00%	4	4	4
b+tree_findK [10]	31.95%	4	12.52	2.15
b+tree_findRangeK [10]	39.51%	4	10.12	2.21
3DConvolution [11]	50.00%	3	6	4.62
bfs_kernel1 [10]	53.37%	4	7.47	4.91
backprop_forward [10]	75.00%	2	2.67	2.89
2DConvolution [11]	90.01%	4	4.44	4.33
backprop_adjust_weights [10]	100.00%	3	3	3.21
bfs_kernel2 [10]	100.00%	1	1	1
GEMM [11]	100.00%	4	4	2.51



(a) Base workload



(b) Real workload

FIGURE 7. Relation between `degree of coalescing` and L1 hit rate.

B. RELATIONSHIP BETWEEN L1 CACHE AND COALESCING

This study analyzes our profiling result with L1 cache in respect of hit rate and sector per request, which is dynamically profiled from NVIDIA Nsight compute [9]. For the experiment, this paper uses 2 criteria, `degree of coalescing` and `estimated sector access`, to compare the result of our profiling result. (refer to Table 3 and Figure 7). As `degree of coalescing` is the criteria that shows data locality of workload, our study compares the result with L1 cache hit rate. 7 To see how many sectors that warp accessed, `estimated sector access` shows how many sectors will be accessed based on the `degree of coalescing` result. This result is compared with L1 sector per request which is extracted from Nvidia Nsight Compute and the result is in Figure 8.

In the case of the stride workload, threads in a warp access data by skipping over n memory locations, showing that data reuse and coalescing are less frequent in this workload. In stride-32, 32 threads in the same warp do not share any of the data between coalescing range. On the other hand, in the case of stride-4 workload, threads access data by skipping over 4 memory locations. This result shows that 8 data locations are re-accessed within coalescing range. This means that the stride-4 workload exhibits a higher `degree of coalescing` than the stride-32 workload. Analysis using the Nsight Compute profiler [9] provided by NVIDIA reveals that the L1 cache hit is higher in the stride-4 workload. Thus, coalescing, as well as access of the same data, affect the L1 cache.

The same_location and coalescing workloads, exhibit a high degree of data reuse and coalescing. The same_location workload allows all threads within a warp to access each data address. Therefore, the `degree of coalescing` for this workload is 100%. In the coalescing workload, the 32 threads in each warp access data determined by their thread IDs. Therefore, all threads in a warp access different data addresses in this workload. However, since the range of the data access falls within 128 bytes, the `degree of coalescing` and the L1 cache hit rate are identical to those of the same_location workload. This demonstrates that the `degree of coalescing` reflects the patterns of memory addresses accessed by the threads. However, an 100% `degree of coalescing` does not necessarily imply an 100% cache hit rate. The latter depends on data locality between thread blocks and the number of data access requests, as well as coalescing information within the warp level.

Figure 7(b) and Table 3 depicts the `degree of coalescing` and L1 cache hit rate of the rodinia [10] and polybench [11] workload. There are some workloads that shows this result including same_location, coalescing, 2DConvolution and so on. The result of those workloads shows that the L1 cache hit rate is observed to exceed 60% when the `degree of coalescing` exceeds 70%. This result shows that workloads with high degrees of coalescing

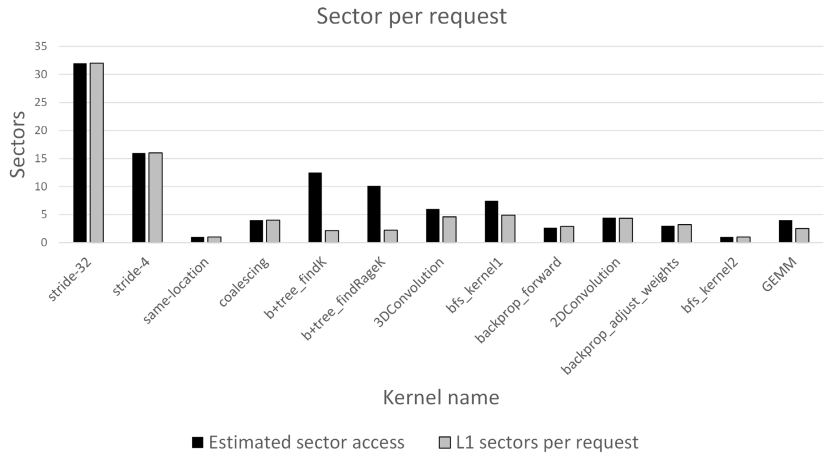


FIGURE 8. Sectors needed for each global load request of workloads.

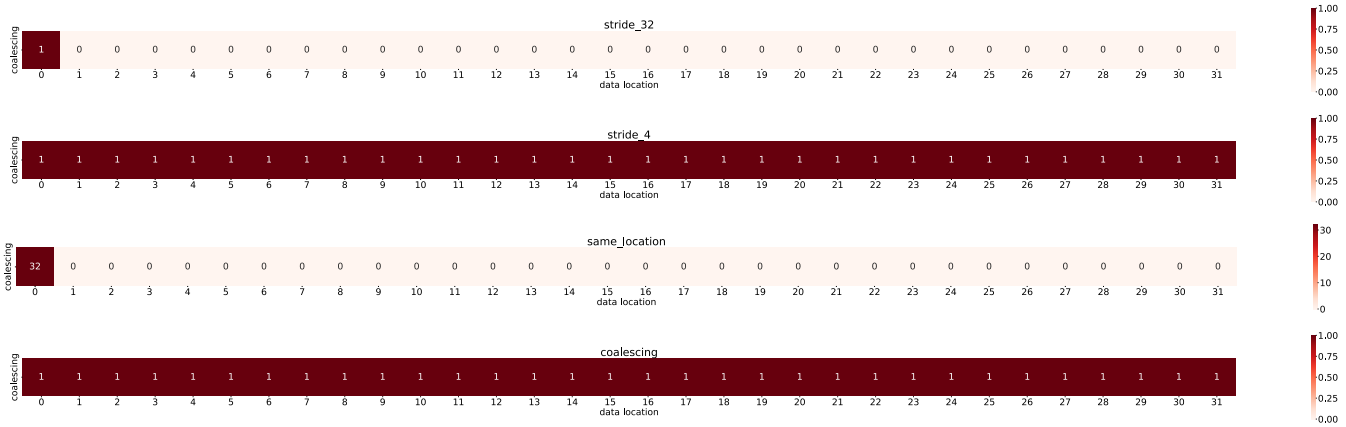


FIGURE 9. Coalescing graph in base workload.

also exhibit high L1 cache hit rates. In the case of B+tree, the findK function and findRageK function exhibit degrees of coalescing of 32.66% and 40.63%, respectively, which are considered low. Their cache hit rates are 55.56% and 69.73%, respectively, which are higher than corresponding degrees of coalescing. This may be attributed to the dense distribution of the data. When comparing the workloads with multiple kernels, for the same workloads, if 1 kernel's `degree of coalescing` is higher than the other's, the L1 hit ratio is also higher. This tendency is shown all of the multi kernel workloads, which is b+tree, bfs and backprop.

As depicted in Figure 8, the L1 sector/request is 2.14 for the findK function and 2.08 for the findRangeK function, indicating that the number of sectors accessed is less than the optimal sector access number, which is 4. Analysis of Figure 10(a) and Figure 10(b) reveals that accesses are concentrated in the first sector. The actual number of sectors accessed can be less than or equal to 4 if data access is well concentrated or coalesced. This result shows that a relatively high cache hit rate is achieved even though the `degree of coalescing` is low due to the b+tree characteristic of accessing specific data repeatedly. This tendency can also be found in other workloads.

C. ANALYSIS THROUGH DEGREE OF COALESCING GRAPH

Based on Figure 9, 10 the formation of the `degree of coalescing` and the data access density within a warp can be visualized. Also, the number of sector accesses' estimation per warp can be deduced from the `degree of coalescing` and coalescing graph. The `sector access expectation` is calculated by multiplying the number of sector accesses within the 128-byte range with the reciprocal of the `degree of coalescing`.

- `num_sectors`: number of sector accesses within 128 bytes

$$sector\_access\_expectation = \frac{num\_sectors}{degree\_of\_coalescing} \tag{2}$$

As mentioned in the NVIDIA profiling guide [9], the number of sectors/requests can be reduced to less than or

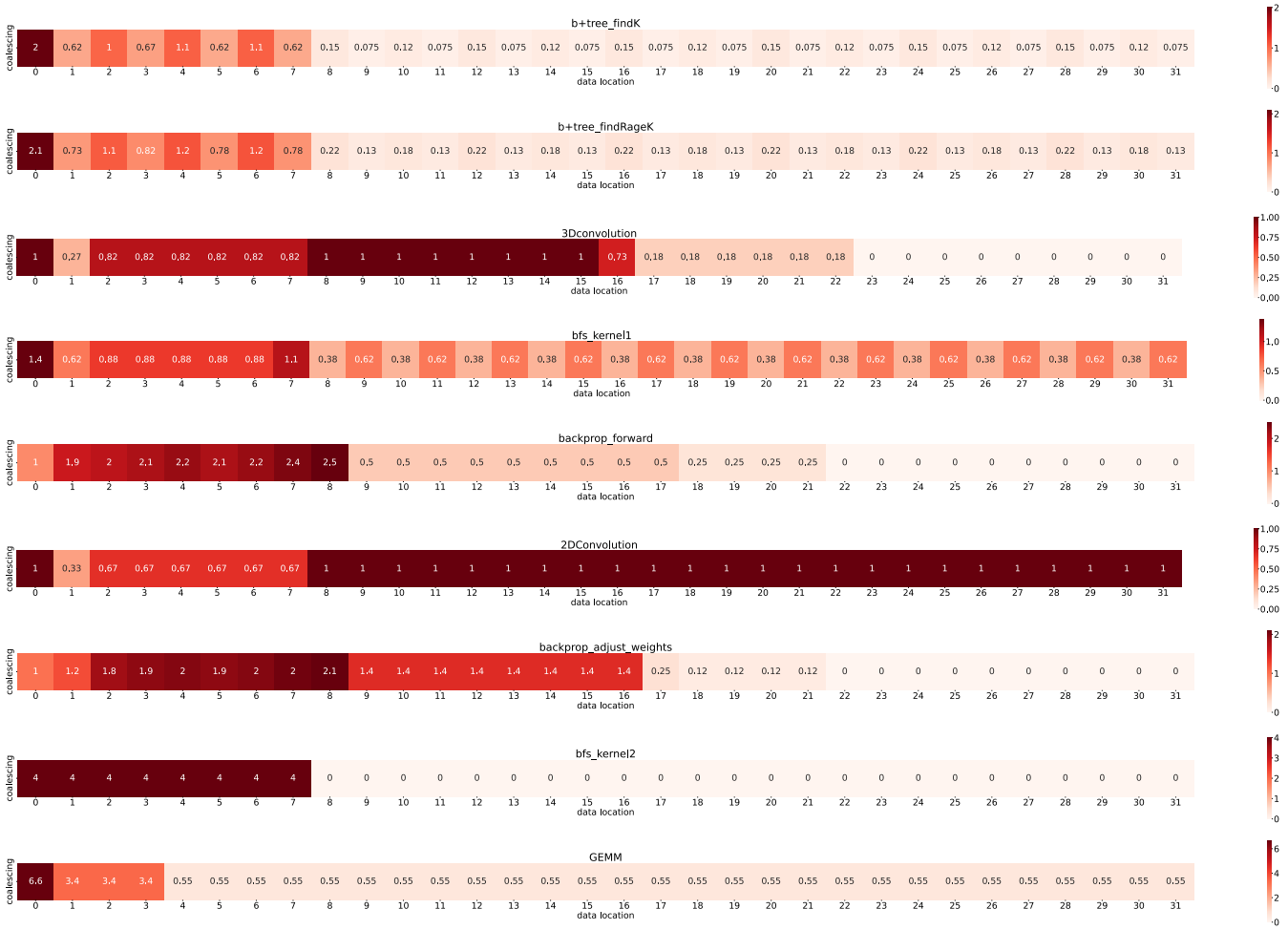


FIGURE 10. Degree of coalescing graph in real workload.

equal to the optimal number, 4, when threads within a single warp has dense or coalesced access. As depicted in Figure 9. 10 and Table 3, the number of sectors/requests is less than or equal to 4 for workloads with high data access density within the warp, e.g., same-location, backprop_forward, and bfs_kernel2 among the executed workloads.

This predicts the number of sector accesses by accounting for the number of sector accesses in the coalescing range of the warp and the proportion of such 128-byte ranges required by calculating the `degree of coalescing` inversely. The predicted value differs from the actual number of L1 sector per request owing to the presence of multiple global memory access commands in the workload with varying degrees of coalescing and densities.

However, in the case of the base workloads, the predicted number of sector accesses and the actual number of L1 sectors per request obtained via profiling using NVIDIA Nsight Compute [9] are identical because the same type of global memory access is performed for all base workloads. As depicted in Figure 10 among the actual workloads, backprop-forward, 2DConvolution and bfs-kernel2 exhibit errors less than 1 sector between the expected number of sector accesses and the actual number of L1 sectors per request.

In the case of the 3DConvolution and bfs-kernel1 workloads, where more than 4 sectors are accessed, the profiling result is consistent with the fact that more than 4 sectors are accessed. These results indicate that the `degree of coalescing` is correlated with the L1 cache hit result. The number of sectors accessed within a warp, as calculated using the `degree of coalescing` and the coalescing graph, reveals that the `degree of coalescing` and the actual number of L1 sectors per request share a meaningful relationship.

V. CONCLUSION

This paper investigates data locality in GPU workloads via PTX code analysis. This study conducted data locality analysis in warp level and focused on profiling the `degree of coalescing` and the `data access density`. We determined the `degree of coalescing` for threads within a warp by identifying the data address accessed by a thread based on the PTX code. Our proposed profiling method also shows that `sector access`

expectation, based on the degree of coalescing and sector accesses per warp, were highly correlated with the actual number of sectors per request. Moreover, our static profiling method visualize the degree of coalescing to represent the characteristics of the workloads. Of the 6 rodinia [10] and 3 polybench [11] workloads executed, 6 workloads exhibited the degree of coalescing exceeding 70% and an L1 cache hit rate exceeding 60%. This result confirms that the L1 cache hit rate tends to be high when the degree of coalescing is high.

In future research, extending this study to develop a scheduler that co-schedules workloads with different patterns of data access is probable. Also, as this research is limited to static profiling, extending profiling method to machine learning based profiling can improve profiling. Using this, we expect to devise multi-workload scheduling that improves L1 cache efficiency.

REFERENCES

[1] A. Li, S. L. Song, W. Liu, X. Liu, A. Kumar, and H. Corporaal, "Locality-aware CTA clustering for modern GPUs," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 297–311, May 2017.

[2] C. Li, S. L. Song, H. Dai, A. Sidelnik, S. K. S. Hari, and H. Zhou, "Locality-driven dynamic GPU cache bypassing," in *Proc. 29th ACM Int. Conf. Supercomputing*, Jun. 2015, pp. 67–77.

[3] S. Lal, B. S. Varma, and B. Juurlink, "A quantitative study of locality in GPU caches for memory-divergent workloads," *Int. J. Parallel Program.*, vol. 50, no. 2, pp. 189–216, Apr. 2022.

[4] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A locality-aware memory hierarchy for energy-efficient GPU architectures," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2013, pp. 86–98.

[5] N. Vijaykumar, E. Ebrahimi, K. Hsieh, P. B. Gibbons, and O. Mutlu, "The locality descriptor: A holistic cross-layer abstraction to express data locality in GPUs," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 829–842.

[6] *CUDA Refresher: The CUDA Programming Model*. Accessed: Jun. 26, 2020. [Online]. Available: https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model

[7] *CUDA Toolkit Document*. Accessed: May 20, 2021. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[8] *BEST Practices Guide: CUDA Toolkit Documentation*. Accessed: May 20, 2021. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#device-memory-spaces

[9] *Nsight Compute Profiling Tool*. Accessed: May 20, 2021. [Online]. Available: https://docs.nvidia.com/nsight-compute/ProfilingGuide/

[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54.

[11] (2012). *Polybench: The Polyhedral Benchmark Suite*. [Online]. Available: http://www.cs.ucla.edu/pouchet/software/polybench

[12] D. Tripathy, A. Abdolrashidi, L. N. Bhuyan, L. Zhou, and D. Wong, "PAVER: Locality graph-based thread block scheduling for GPUs," *ACM Trans. Archit. Code Optim. (TACO)*, vol. 18, no. 3, pp. 1–26, 2021.

[13] D. Tripathy, A. Abdolrashidi, Q. Fan, D. Wong, and M. Satpathy, "LocalityGuru: A PTX analyzer for extracting thread block-level locality in GPGPUs," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Oct. 2021, pp. 1–8.

[14] X. Tang, A. Pattnaik, O. Kayiran, A. Jog, M. T. Kandemir, and C. Das, "Quantifying data locality in dynamic parallelism in GPUs," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, pp. 1–24, 2018.

[15] S. Sethumurugan, J. Yin, and J. Sartori, "Designing a cost-effective cache replacement policy using machine learning," in *Proc. IEEE Int. Symp. High-Performance Comput. Archit. (HPCA)*, Feb. 2021, pp. 291–303.

[16] Z. Shi, X. Huang, A. Jain, and C. Lin, "Applying deep learning to the cache replacement problem," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 413–425.

[17] L. V. Rodriguez, F. Yusuf, S. Lyons, E. Paz, R. Rangaswami, J. Liu, M. Zhao, and G. Narasimhan, "Learning cache replacement with CACHEUS," in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 341–354.

[18] F. Hong, J. Zhang, and X. Yuan, "Access pattern learning with long short-term memory for parallel particle tracing," in *Proc. IEEE Pacific Visualizat. Symp. (PacificVis)*, Apr. 2018, pp. 76–85.

[19] T. Tang, X. Yang, and Y. Lin, "Cache miss analysis for GPU programs based on stack distance profile," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.*, Jun. 2011, pp. 623–634.

[20] C. Nugteren, G.-J. van den Braak, H. Corporaal, and H. Bal, "A detailed GPU cache model based on reuse distance theory," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 37–48.

JIEUN KIM received the B.S. degree in law and computer science from Sookmyung Women's University, in 2020, and the M.S. degree from the Department of Computer Science and Engineering, Seoul National University, in 2023. Her research interests include GPU workload profiling and GPU in deep learning.

HYEONSANG EOM received the B.S. degree in computer science and statistics from Seoul National University (SNU), Seoul, South Korea, in 1992, and the M.S. and Ph.D. degrees in computer science from the University of Maryland at College Park, MD, USA, in 1996 and 2003, respectively. He was an Intern with the Data Engineering Group, Sun Microsystems, CA, USA, in 1997, and a Senior Engineer with the Telecommunication Research and Development Center, Samsung Electronics, South Korea, from 2003 to 2004. He is currently a Professor with the Department of Computer Science and Engineering, SNU, where he has been a Faculty Member, since 2005. His research interests include high performance storage systems, operating systems, distributed systems, cloud computing, energy efficient systems, fault-tolerant systems, security, and information dynamics.

YOONHEE KIM received the bachelor's degree from Sookmyung Women's University, in 1991, and the master's and Ph.D. degrees from Syracuse University, in 1996 and 2001, respectively. She is currently a Professor with the Computer Science Department, Sookmyung Women's University. She was a Research Staff Member with the Electronics and Telecommunication Research Institute, in 1991 and 1994. Before joining as a Faculty Member of Sookmyung Women's University, in 2001, she was a Faculty Member of the Computer Engineering Department, Rochester Institute of Technology, NY, USA. Her research interest includes many aspects of runtime support and management in distributed computing systems. She is a member of OGF. She has served on variety for program committees, advisory boards, and editorial boards.