Scheduling-Aware Prefetching: Enabling the PCIe SSD to Extend the Global Memory of GPU Device

Tse-Yuan Wang*†, Chun-Feng Wu*†, Che-Wei Tsao*†, Yuan-Hao Chang† and Tei-Wei Kuo*‡§
*Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
†Institute of Information Science, Academia Sinica, Taiwan
‡College of Engineering, City University of Hong Kong, Hong Kong
§NTU High Performance and Scientific Computing Center, National Taiwan University, Taiwan
E-mail: {tseyuan20, cfwu, johnson}@iis.sinica.edu.tw, bearman.sky@gmail.com, ktw@csie.ntu.edu.tw

***Abstract*—The evolution of Cyber-Physical Systems (CPSs) and Internet of Things (IoTs) enables mobile and smart embedded devices to be equipped with embedded GPUs for accelerating data-intensive applications. To cut down device prices and reduce energy consumption, current GPUs adopt the unified memory architecture to extend memory size with using the PCIe SSD which is cheaper than directly enlarging the off-chip DRAM on the GPU. However, adopting the unified memory architecture, data shall be moved to the host DRAM before being moved to the off-chip DRAM and thus it leads to serious contention issues among CPUs and GPUs on the host DRAM. Although the advent of new communication technology provides the opportunity for GPUs to directly access the PCIe SSD without passing the host DRAM, it leads to high data movement costs because the latency gap between the off-chip DRAM and the PCIe SSD is large. To enhance the performance of the low-cost energy-efficient GPU memory systems, this work advocates a hardware-controller-based memory extension solution to not only avoid the contention issues on the host DRAM but also reduce the data movement costs. Particularly, we propose a scheduling-aware prefetching design to perform data prefetching by utilizing the information from the hardware warp scheduler. The proposed solution was evaluated by a series of intensive experiments and the results are encouraging.**

I. Introduction

The evolution of Cyber-Physical Systems (CPSs) and Internet of Things (IoTs) enables mobile and smart embedded devices to be equipped with embedded GPUs for accelerating data-intensive applications, such as computer graphic, neural networks (NNs) [1], and machine learning applications [2]. Aiming at accelerating these applications, some devices are equipped with embedded Graphics Processing Units (GPUs) [3], [4], [5]. In order to cut down the price and reduce the energy consumption, embedded GPUs are usually equipped with smaller off-chip dynamic random-access memory (DRAM). In contrast to previous GPU memory size which is limited by the size of off-chip DRAM, current GPUs provide the unified memory architecture to unify memory space provided by the off-chip DRAM and the host DRAM. That is, GPUs are able to access the data resided in the host DRAM so as to extend the memory space. Moreover, with the help of memory swapping, GPUs can also access the data stored in swap devices after the data is swapped back the host DRAM. To be noted, current mobile devices are usually equipped with Peripheral Component Interconnect Express (PCIe) Solid-State Drives (SSDs) which can be configured as a swap device. Under the unified memory architecture, host DRAM shared between GPU and Central Processing Units (CPUs) suffers from serious memory contention issues especially when the mobile devices and smart embedded devices are equipped with limited host DRAM. Fortunately, due to the evolution of the wire-based communications protocol, it provides the opportunity to enable GPUs to access swap devices without moving the data to the host DRAM in advance. However, directly accessing the data on the swap device leads to serious performance degradation because the performance gap between the off-chip GPU memory and the swap device is large. *Moreover, in contrast to CPU tasks, GPU tasks show weaker temporal locality and thus the management of host DRAM meets the challenge of fast locality saturation.* That is, the effectiveness on capturing process locality saturates quickly if the host DRAM is managed by merely using page replacement approaches. Such a contradiction motivates us to look for solutions to enable an efficient low-cost energy-efficient GPU memory extension systems.

To mine the information behind large amounts of data retrieved from sensors and social media, data-intensive applications are widely used in this few years. To efficiently run the data-intensive applications, large amounts of data shall be placed in the memory devices, such as DRAM, before being accessed by CPUs or GPUs. However, extending DRAM size not only increases the prices but leads to serious energy consumption. Aiming at providing low-cost memory space, persistent memory researches can be classified into two classes, that is the unified memory and memory extension solutions. Non-Volatile Random Access Memory (NVRAM), e.g., Phase Change Memory (PCM) and STT-RAM, is regarded as a possible unified memory solution for taking the place of DRAM. However, until now, NVRAM is still in the testing or even the development phases due to the manufacturing issues in terms of cost and maturity [6], [7], [8]. On the other hand, the memory extension solutions aiming at incorporating different memory devices are production-ready. Based on different managements on performing data movements between different memory devices, the memory extension solutions can be classified into software-controller-based [9] and hardware-controller-based memory extension solutions [10]. To be specific, for software-controller-based and hardware-controller-based memory extension solutions, data movements are performed by the operating system (OS) and the micro-controller (or hardware circuit) respectively. Virtual memory management is a well-known software-controller-based memory extension solution and it works by performing data swapping between the host DRAM and swap devices with the help of operating systems. However, all data shall be moved to the host DRAM before being accessed and thus the memory contention issues become more serious when both CPUs and GPUs share the host DRAM. On the other hand, hardware-controller-based memory extension solutions introduce a micro-controller which helps to move the data between different memory devices without passing the host DRAM. Aiming at avoiding the contentions on host DRAM, this work focuses on proposing a hardware-controller-based memory extension solution.

The main challenge of the hardware-controller-based memory extension solution is that the data movement costs between different memory devices are expensive especially when the performance gap between the memory devices is large. To alleviate the data movement costs by reducing the frequency of the data movements, some of the micro-controller designs monitor and keep the data which has higher reuse probability in the faster memory with taking advantage of the process access behaviors. Due to the fact that CPU tasks usually show strong temporal access locality and thus recently accessed data has higher probability to be accessed in the near future. Several well-known replacement designs (such as ARC [11], CAR [12], and CLOCK-Pro [13]) focus on identifying and avoiding evicting the recently accessed data in the faster memory with lower management overhead. To further eliminate the data movement costs, data prefetching is a more aggressive approach to predict which data will be accessed recently and fetch the predicted data from slower devices to faster devices. To accurately predict the data, some researchers rely on the process access behavior. For example, researchers show that CPU tasks show strong spatial and temporal locality on the virtual addressing space [10] and thus propose the page-on-page prefetching strategy to fetch the data pages with virtual addressed nearby the one which incurs a page fault. However, due to different design rational behind CPU and GPU tasks, GPU tasks are usually designed for processing large amount of independent data so as to fully utilize the high parallelism exposed by GPUs. Therefore, GPU tasks usually show low temporal locality and previous approaches designed for capturing temporal locality become in-effective on dealing with GPU tasks.

Aiming for accelerating data-intensive applications, mobile and smart embedded devices are usually equipped with embedded GPUs. To cut down device prices and reduce energy consumption, current GPUs adopt the unified memory architecture to extend memory size with using the PCIe SSD which is cheaper than directly enlarging the off-chip DRAM on the GPU. However, based on the unified memory architecture, host DRAM is shared between CPUs and GPUs, and thus it leads to serious contention issues on the host DRAM. Although GPUs can directly access the PCIe SSD without passing the host DRAM, it leads to high data movement costs because the latency gap between the off-chip DRAM in the GPU and the PCIe SSD is large. In contrast to previous works, we propose a Scheduling-Aware Prefetching (SAP) design to enhance the performance of the low-cost energy-efficient GPU memory systems by (1) advocating a hardware-controller-based memory extension solution to extend GPU memory with PCIe SSD and thus avoid the memory contention issues on host DRAM, and (2) utilizing the information of internal hardware warp scheduler inside GPU device to perform data prefetching. A series of experiments are conducted to evaluate the performance of the proposed SAP design, and the results show that the proposed SAP design could reduce up to 99% of effective access time compared with the investigated well-known page replacement algorithms.

The rest of this paper is organized as follows. Section II presents the background, observation and motivation of this work. In Section III, a Scheduling-Aware Prefetching design is proposed to enhance the access performance of memory system. The experimental results are then reported and discussed in Section IV. Section V concludes this work.

II. Background, Observation and Motivation

A. Background: Low-Cost Energy-Efficient GPU Memory Extension

GPUs are widely used to accelerate data-intensive applications, such as computer graphic and deep learning, and thus it usually has large memory demands for holding the data. To meet the memory demands, GPUs are usually equipped with an off-chip DRAM also called the global memory. In addition to the off-chip memory, in recent years, NVIDIA proposed the Unified Memory Architecture [14] to allow GPUs to access the memory on the host systems, so that the total accessible memory space can be further extended. However, DRAM suffers from several shortcomings especially when its size is increased, such as the expensive unit costs and high standby power [15], and these shortcomings seriously increase the costs of GPU, in terms of both prices and energy consumption. Fortunately, in recent years, the manufacturing technology of NAND-flash chips becomes sophisticated and thus several low-price and high bandwidth NAND-flash based devices hit the markets, such as SSD and NVDIMM [10]. The difference of prices between DRAM and NAND-flash memory is shown in Table I. On the other hand, the communication interface (e.g., PCIe) has rapidly developed to provide higher bandwidth for holding the huge data movements between devices and the host system. Under this trend, several researches advocate to extend the main memory on the host system by using the NAND-flash based storage devices, such as PCIe SSD, so as to provide larger memory space with and cheaper unit costs and nearly-zero standby power [16], [17].

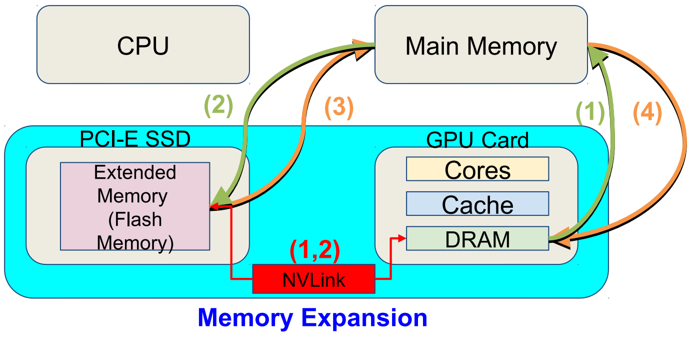
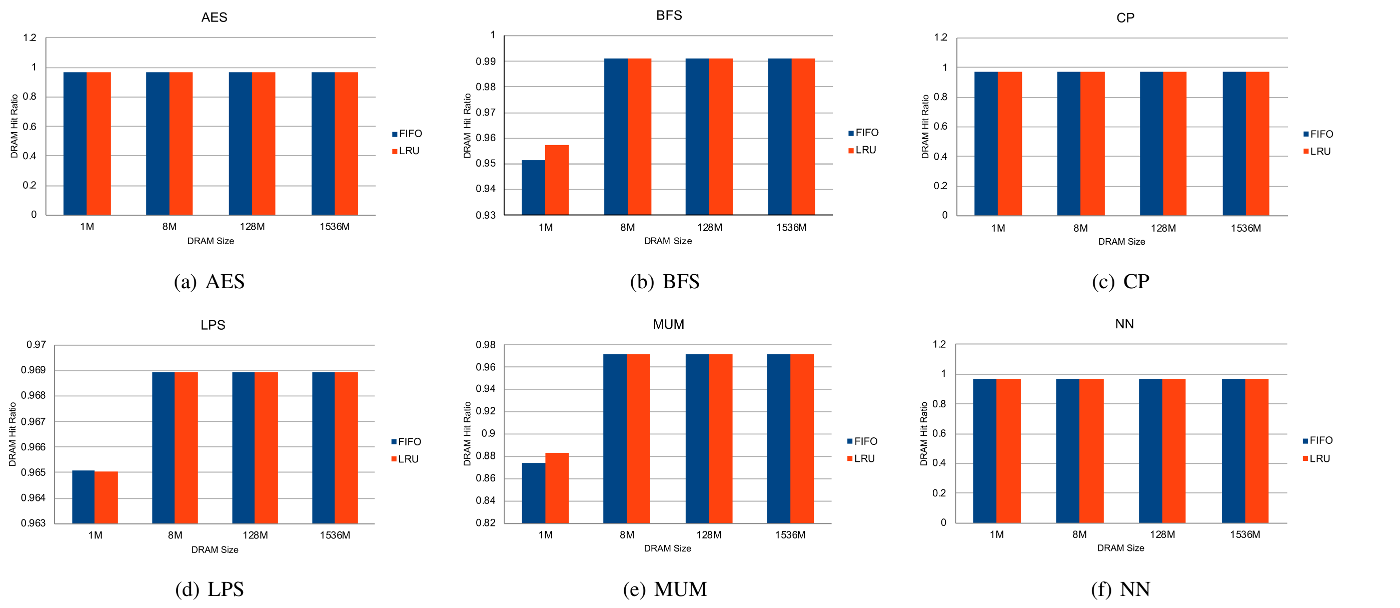


Fig. 1. Data Movement among Host's RAM, GPU Device's RAM and PCIe SSD

Aiming at expanding the GPU memory space with cheaper prices, a *swap-like protocol* is adopted to enable storage devices (e.g., PCIe SSD) as a part of GPU memory. To hide this complex memory hierarchy from GPU programmers, host memory is used to buffer the data between GPU and host system so that GPU programmers only need to interact with host memory. For example, a *cudaMemcpy* Application Programming Interface (API) in the Compute Unified Device Architecture (CUDA) library is provided by NVIDIA for programmers to decide the data movements between the GPU off-chip memory and the host memory. On the host system, when the host memory suffers high memory pressure, the operating systems help to perform the data movements by swapping the data between host memory and swap devices. The design rationale behind the swap-like protocol, as shown in Figure 1, is to migrate the to-be-accessed data from the host memory to the off-chip GPU memory once the data is requested, and the data shall be first moved to the host memory if it is resided on the external devices as illustrated in step (1) to (4) in Figure 1. Specifically, when the memory space inside the GPU device is not sufficient, some selected data decided by the programmer code will be evicted to the host system, and the host system will allocate a memory space for this data, as shown in step (1). If there is no enough memory space on the host system, some cold data will be replaced and swapped to storage devices by the operating system, as shown in step (2). On the other hand, if the data required by the GPU is resided in the storage device, this data will be first moved to the host memory, and then moved from the host memory to the GPU device, as shown in steps (3) and (4). Although this protocol provides the opportunity for expanding the GPU memory with the storage devices, it leads to the intensive host-memory contention issue between CPU and GPU. Fortunately, NVLink [18], a new communication technology, can be adopted to enable GPU to access the data on the storage device directly without buffering data in the host memory in advance, as show in steps (1,2), so that the memory contention issue can be eliminated. NVLink adopts the technology of Remote Directly Memory Access (RDMA) to enable data transmissions between two PCIe devices, that is GPUs and PCIe SSDs in our case, without using host memory. However, frequent data movements between GPU and storage devices seriously degrade system performance because the



(a) AES (b) BFS (c) CP

(d) LPS (e) MUM (f) NN

Fig. 2. DRAM Hit Ratio under Different Page Replacement Approaches

access latency of the flash-based storage devices are several orders slower than the off-chip GPU DRAM, as shown in Table I.

TABLE I
THE CHARACTERISTICS OF DRAM AND FLASH MEMORY.

Type	DRAM	Flash SLC [19]	Flash MLC$_{\times 2}$ [20]
Price (USD/GB, 2017)	6.79	3.05	0.52
Serial Access	$1ns$	$10ns$	$20ns$
Random Read	$60ns$	$50\mu s$	$75\mu s$
Write/Program	$60ns$	$550\mu s$	$1300\mu s$
Erase	NA	$1.5ms$	$3.8ms$

B. Observation: Fast Locality Saturation

To improve system performance by reducing the amounts of data movements, page replacement approaches are used to hold the most frequently accessed data in the memory with taking advantage of process locality. According to several previous works [11], [12], [13], most of the page replacement approaches monitor and capture the access localities of memory pages by utilizing the recorded access information, such as access recency and frequency. However, in contrast to a CPU, which is a general-purpose processor with few complex computing cores and designed for running few serial tasks, a GPU is a special-purpose processor with huge amounts of simple computing cores and is designed for running plenty of data-parallelism tasks. That is, in each execution batch, most GPU cores run the same instruction on processing different data. Specifically, GPU adopts the single-instruction multiple-thread (SIMT) architecture and that is several threads running the same codes are binded into a batch or warp [21], [22].

In most cases, the relation between two warps is independent and thus tasks run in GPU usually show weak temporal locality. In this case, for GPU-based systems, page replacement approaches are effective only on capturing the spatial locality exposed from the data and it is hard to capture the temporal behaviors of GPU tasks. Based on our observations, applying page replacement approaches on GPU tasks usually suffers from the issue of fast locality saturation. That is, the effectiveness on capturing process locality saturates quickly and thus, even when larger DRAM is equipped in the GPU, the DRAM hit ratio cannot be further improved.

To validate the observed fast locality saturation of GPU tasks, we analyze six task behaviors collected from the modified GPGPU-Sim [23] simulator. All tasks are selected from the representative benchmarks in ISPASS [23]. In the experiments, we simulated two memory layers, L2 cache and main memory (DRAM or global memory), and the configurations are set depending on NVIDIA GTX480. That is, the size of the L2 cache is set to 768 KB. We evaluate two representative page replacement approaches, such as the first-in-first-out (FIFO) and the least-recently-used (LRU), on improving the DRAM hit ratio under different setting of DRAM size, that is from 1MB to 1.5GB. Figure 2 shows the evaluation results, where the x-axis shows the DRAM size and the y-axis indicates the DRAM hit ratio. The fast locality saturation phenomenon is clearly shown in all benchmarks and that is the DRAM hit ratio approaches to saturation with setting the DRAM to merely 8 MB.

C. Motivation

According to our observations, adopting unified memory architecture to extend GPU memory space with using PCIe SSD may incur serious contentions on host DRAM especially when the embedded systems are equipped with limited host DRAM. Although GPUs can directly access PCIe SSD without moving the data to host DRAM beforehand, it leads to serious performance degradation due to the large performance gap between the PCIe SSD and the off-chip DRAM in the GPUs. In contrast to previous works, this work focuses on the solution that can not only avoid the contention issues on the host DRAM but also improve system performance by prefetching the data with considering process information provided by the GPU warp scheduler. The technical challenges fall on (1) how to design a hardware controller to avoid the memory contention issue and (2) how to exploit the process information in the warp scheduler to perform data prefetching.

III. SCHEDULING-AWARE PREFETCHING

A. Design Overview

In the typical modern system architecture, the GPU device is regarded as a co-processor in the host to accelerate specific applications such as graphic processing, deep learning, etc. The host usually has a storage device (block device) as the swap device, which is used to extend the main memory space to provide the extra memory space when the DRAM space of the host is not sufficient. This extended memory space can also provide to the GPU device, which can indirectly use this extended memory space by the host when the GPU device's main memory space is also not sufficient. Fortunately, by RDMA technology, we can use a PCIe SSD as the minor main memory for the GPU device to extend the main memory space. The complex data movement operation among the RAM of the host, the RAM of the GPU device, and the PCIe SSD can be efficiently avoided. Without significantly modifying the software stack of CUDA programming, and completely utilizing the information of the internal hardware warp scheduler, we design a new hardware component "Memory Manager" to collect the warp information, and the detail of this component is described in Section III-B. The complete system architecture is shown in Figure 3.

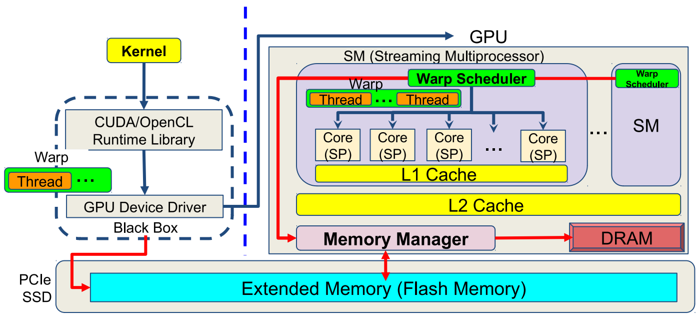


Fig. 3. System Architecture

B. *Internal GPU device information – Warp Scheduling*

The typical GPU program can be divided into two parts, CPU execution part and GPU execution part. In the former, it usually contains data allocation, GPU device initialization, etc. In the latter, it is usually called "kernel," and which is usually the sub-programs or functions of the GPU program. For reducing the complexity of writing the GPU program, many runtime libraries (such as CUDA and OpenCL runtime library) are proposed to provide the basic functions, and the GPU device driver also provides the interface for utilizing the hardware resource. The kernel will be divided into many threads by the previous two layers, and threads will be grouped into warps according to the hardware configuration of the GPU device. The threads in the same warp usually perform the instruction with the same type (operation, memory access, etc.), and it will be performed together. GPU is the data parallelism model, and data might be assigned into any one of GPU cores. The hardware warp schedulers decide the execution order of instruction. Profiling the kernel might obtain the memory access information, but the internal hardware warp scheduler in the GPU device might reschedule the warps, and the memory access patterns might be changed and can not be predicted. Therefore, using the information of internal hardware warp schedulers can more accurately predict the access behavior for the future.

For reducing the complexity of writing the GPU program and utilizing the information of the hardware warp schedulers, we proposed a new component "Memory Manager" inside the GPU device to fully utilize the information of hardware schedulers to dynamically data movement without changing the writing way of GPU program. The instructions of warp have many types. We only consider the instructions about memory access operations, such as memory load and store instructions, because the data should be loaded/stored from/to main memory before/after the data are operated. Therefore,

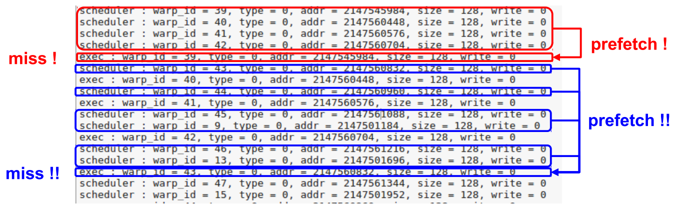

```
scheduler : warp_id = 39, type = 0, addr = 2147545984, size = 128, write = 0
scheduler : warp_id = 40, type = 0, addr = 2147560448, size = 128, write = 0
scheduler : warp_id = 41, type = 0, addr = 2147560576, size = 128, write = 0
scheduler : warp_id = 42, type = 0, addr = 2147560704, size = 128, write = 0
exec : warp_id = 39, type = 0, addr = 2147545984, size = 128, write = 0
scheduler : warp_id = 43, type = 0, addr = 2147560832, size = 128, write = 0
exec : warp_id = 40, type = 0, addr = 2147560448, size = 128, write = 0
scheduler : warp_id = 44, type = 0, addr = 2147560960, size = 128, write = 0
exec : warp_id = 41, type = 0, addr = 2147560576, size = 128, write = 0
scheduler : warp_id = 45, type = 0, addr = 2147561088, size = 128, write = 0
scheduler : warp_id = 9, type = 0, addr = 2147501184, size = 128, write = 0
exec : warp_id = 42, type = 0, addr = 2147560704, size = 128, write = 0
scheduler : warp_id = 46, type = 0, addr = 2147561216, size = 128, write = 0
scheduler : warp_id = 13, type = 0, addr = 2147501696, size = 128, write = 0
exec : warp_id = 43, type = 0, addr = 2147560832, size = 128, write = 0
scheduler : warp_id = 47, type = 0, addr = 2147561344, size = 128, write = 0
scheduler : warp_id = 15, type = 0, addr = 2147501952, size = 128, write = 0
```

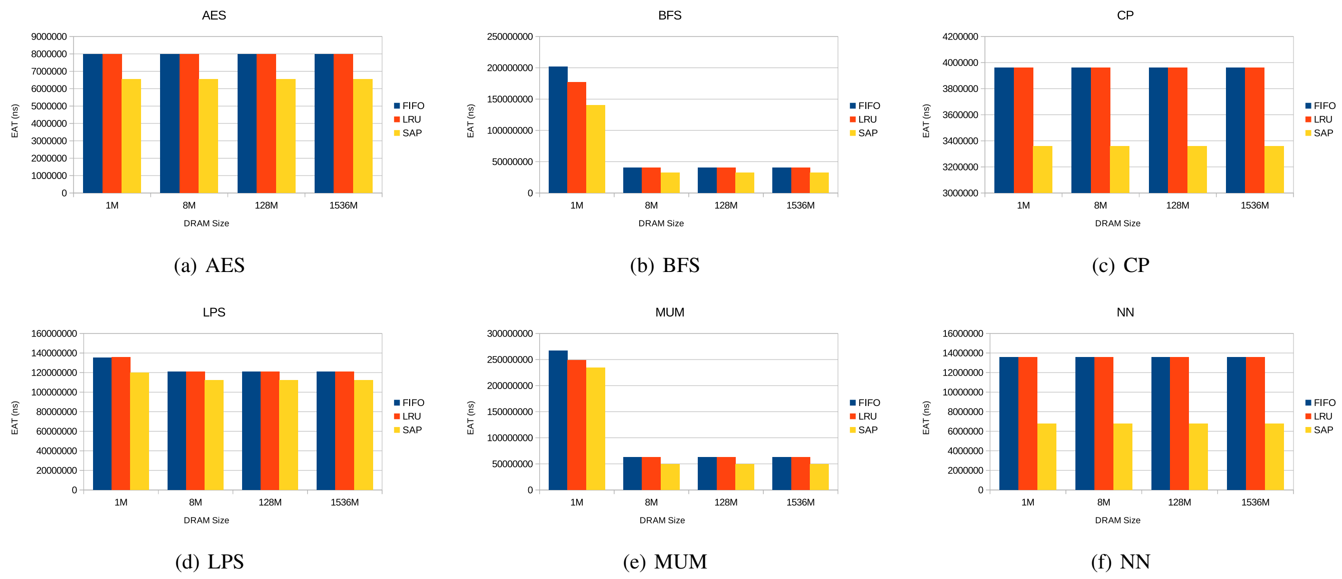
Fig. 4. Scheduler-Aware Prefetching

the memory manager will periodically watch the warp schedulers to obtain the instructions about memory access. If the accessed data location is not L1 or L2 cache, the accessed memory range will be recorded in the memory manager, and the adjacent memory range will be grouped together. Because the access latency of flash memory is very long, the data movement between the GPU device's main memory and the PCIe SSD only occurs when the required accessed data is missing in the GPU device's main memory. PCIe SSD usually has high access bandwidth, so migrating more data at once is suitable and reasonable. The data in the main memory level of the GPU device usually have a strong spatial access locality, so using the large size for the management unit of the main memory in the GPU device is also suitable. In there, the management unit is set to 4KB. When data cache miss in the DRAM of GPU device happens, the memory manager will use the collected warp information about memory access instruction to prefetch data from PCIe SSD to DRAM. If the data of the recorded memory range are already in DRAM, then the corresponding management unit of this memory range will not be prefetched to DRAM. This prefetching operation is based on the information of hardware warp scheduler, so we called it "Scheduler-Aware Prefetching (SAP)," and the procedure of SAP can be illustrated by Figure 4. In there, the red box is to illustrate the first prefetching. Before the first data cache miss occurs, the information of warp scheduler is recorded, and the memory addresses of recorded memory access instructions are 2147545984, 2147560448, 2147560576, and 2147560704, respectively. In there, the latter 3 items will be grouped together because they are in the same management unit, and the first item will be recorded to other alone items. When the first data cache miss happens, the data of these two management units will be prefetched to the DRAM of the GPU device from PCIe SSD, and this can better utilize the access bandwidth of storage. Therefore, when the following memory access instructions (the accessed memory accesses are 2147560448, 2147560576, and 2147560704) is coming, data are already in the DRAM of the GPU device, and data cache miss can be efficiently avoided. In past research works, CPU usually has the speculative execution technology which is usually efficient and has good performance, but GPU does not adopt this technology and it could not be efficiently executed in GPU, because GPU adopts the data parallelism optimization model. Therefore, we designed the SAP scheme to fully and efficiently utilize the information of the hardware warp scheduler. SAP does not need to spend unnecessary maintenance operations to monitor the access behavior of the management unit compared to the traditional page replacement algorithms for managing CPU-based memory.

IV. Performance Evaluation

A. *Experimental Setup*

We modify GPGPU-Sim [23] to collect the warp information and the memory access trace. A trace-driven simulation is created to simulate the Unified Memory (contained different memory materials) for GPU device with PCIe SSD. The hardware configuration of GPU device uses NVIDIA GTX480 configure, and DRAM size is set from



(a) AES (b) BFS (c) CP

(d) LPS (e) MUM (f) NN

Fig. 5. Effective Access Time by Different Cache Algorithms

1MB to 1.5GB. The characteristics of simulated unified memory are shown in Table II. The access latency of flash memory is very long, so we use the effective access time to evaluate the performance of different cache policies. In there, we use FIFO and LRU algorithms as the comparisons to evaluate our design (SAP), and the management unit size of all policies is 4KB. For evaluating the performance of different policies on various application cases, we use the benchmarks published in ISPASS-2009 paper [23] to evaluate the performance of the proposed design.

TABLE II
THE SPECIFICATION OF EVALUATED UNIFIED MEMORY.

Type	Access Latency
DRAM Access Latency (Read)	60ns
DRAM Access Latency (Write)	60ns
Flash Memory Random Access Latency (Read)	50,000ns
Flash Memory Random Access Latency (Write)	550,000ns
Flash Memory Serial Access Time (ns/Bytes)	5ns

B. Experimental Results

1) Effective Access Time: In computer architecture, effective access time is a famous and common metric to evaluate memory system performance. The access latency of flash memory is very long, so the great cache policy should efficiently keep or pre-fetch that the might be used data in DRAM, or the effective access time of unified memory will be seriously affected. In there, we set the management unit size to 4KB to ensure that the spatial access locality can be efficiently captured by all cache policies. The effective access time on different benchmarks by different cache policies are shown in Figure 5, and in these figures x-axis is to indicate the different DRAM size and y-axis is the effective access time. According to these figures, we can observe that the effective access time is not decreased when DRAM size of the GPU device is over 8 MB by different cache policies. Our design can efficiently outperform FIFO and LRU policies in different DRAM size and benchmarks, and in FIFO and LRU policies still does not has the great cache hit ratio even when DRAM size is large, because the access behavior of management unit in GPU device is not predictable and the access behavior of data is not repeated. Therefore, these experiments strongly demonstrate that using a recording way of the traditional cache policies can not catch up the working set.

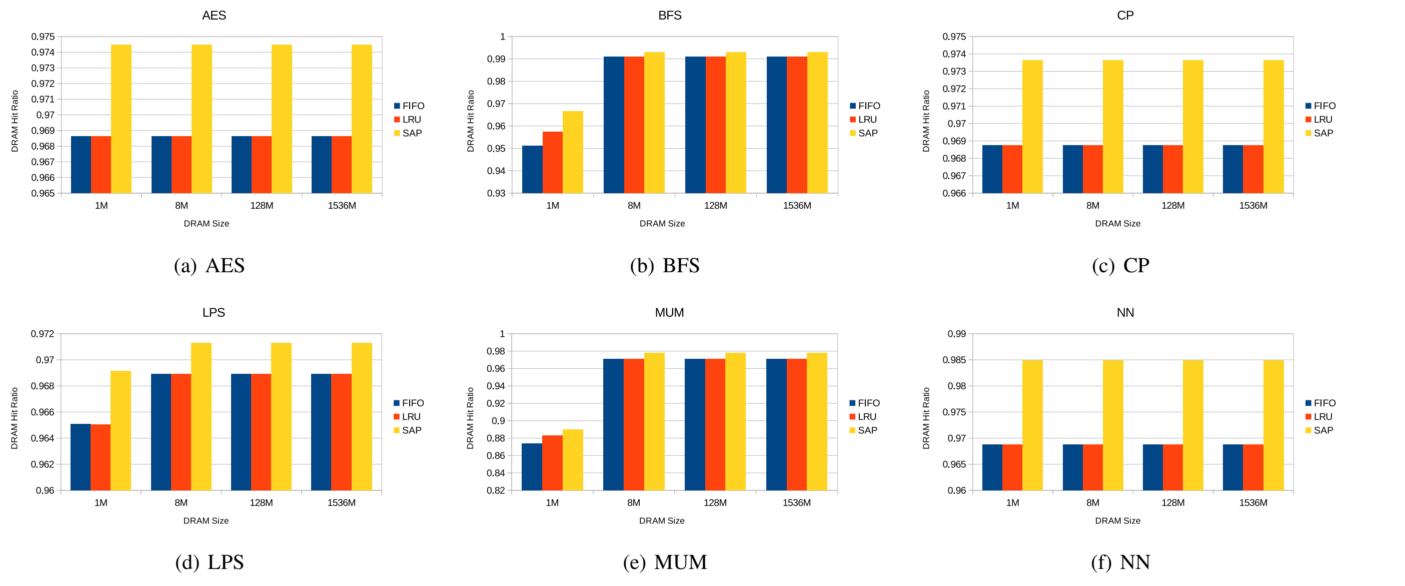
2) DRAM Cache Hit Ratio: For evaluating the performance of cache policies, the most intuitive way is to use cache hit ratio to evaluate. The DRAM cache hit ratio on different benchmarks by different cache policies are shown in Figure 6, and in these figures x-axis is to indicate the different DRAM size and y-axis is the DRAM cache hit ratio. According to these figures, we can observe that FIFO and LRU policies have the same performance in AES, CP, and NN benchmarks even on different DRAM size, and this can demonstrate that most of data does not have strong temporal access locality in main memory level. In BFS and MUM benchmarks, the performance of LRU policy can outperform FIFO policy when DRAM size is 1M, but FIFO and LRU policies also have the same performance when DRAM size is over 8MB, and this also demonstrates that data only have very weak temporal access locality in main memory level. However, in Figure 6 we can observe that the performance of our design can significantly outperform other cache policies, and this strongly demonstrates that the traditional cache policies are not suitably used to manage the main memory of GPU device.

V. Conclusion

Aiming for providing an efficient low-cost energy-efficient GPU memory extension systems, this work advocates a hardware-controller-based memory extension solution to avoid CPUs and GPUs from contending host DRAM and also proposes a scheduler-aware prefetching design to further improve system performance. Particularly, a scheduler-aware prefetching policy exploits the process information provided by warp scheduler so as to predict memory access patterns and perform accurate data prefetching. A series of experiments proves that our design can efficiently improve the access performance of unified memory for GPU device with PCIe SSD.

VI. Acknowledgement

This work was supported in part by Academia Sinica under grant no. AS-GCS-110-08 and AS-CDA-107-M05 and Ministry of Science and Technology under grant nos. 110-2223-E-001-001, 108-2221-E-001-001-MY3, and 108-2221-E-001-004-MY3.



(a) AES (b) BFS (c) CP

(d) LPS (e) MUM (f) NN

Fig. 6. DRAM Hit Ratio by Different Cache Algorithms

References

[1] C.-F. Wu, M.-C. Yang, Y.-H. Chang, and T.-W. Kuo, "Hot-spot suppression for resource-constrained image recognition devices with nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2567–2577, 2018.

[2] Y. T. Ho, C.-F. Wu, M.-C. Yang, T.-Y. Chen, and Y.-H. Chang, "Replanting your forest: Nvm-friendly bagging strategy for random forest," in *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2019, pp. 1–6.

[3] H. Lee, M. Shafique, and M. A. Al Faruque, "Aging-aware workload management on embedded gpu under process variation," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 920–933, 2018.

[4] M. T. Satria, S. Gurumani, W. Zheng, K. P. Tee, A. Koh, P. Yu, K. Rupnow, and D. Chen, "Real-time system-level implementation of a telepresence robot using an embedded gpu platform," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1445–1448.

[5] S. Wang, G. Zhong, and T. Mitra, "Cgpredict: Embedded gpu performance estimation from single-threaded applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–22, 2017.

[6] Y.-W. Kang, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, and S.-Y. Ho, "On minimizing analog variation errors to resolve the scalability issue of reram-based crossbar accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3856–3867, 2020.

[7] J. Lüttgau, M. Kuhn, K. Duwe, Y. Alforov, E. Betke, J. Kunkel, and T. Ludwig, "Survey of storage systems for high-performance computing," *Supercomputing Frontiers and Innovations*, vol. 5, no. 1, pp. 31–58, 2018.

[8] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging nvm: A survey on architectural integration and research challenges," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 2, pp. 1–32, 2017.

[9] C.-F. Wu, Y.-H. Chang, M.-C. Yang, and T.-W. Kuo, "When storage response time catches up with overall context switch overhead, what is next?" *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4266–4277, 2020.

[10] C.-F. Wu, Y.-H. Chang, M.-C. Yang, and T.-W. Kuo, "Joint management of cpu and nvdimm for breaking down the great memory wall," *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 722–733, 2020.

[11] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache," in *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies*, ser. FAST '03. Berkeley, CA, USA: USENIX Association, 2003, pp. 115–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=1090694.1090708

[12] S. Bansal and D. S. Modha, "Car: Clock with adaptive replacement," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, ser. FAST '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 187–200. [Online]. Available: http://dl.acm.org/citation.cfm?id=1096673.1096699

[13] S. Jiang, F. Chen, and X. Zhang, "Clock-pro: An effective improvement of the clock replacement," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 35–35. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247395

[14] W. Li, G. Jin, X. Cui, and S. See, "An evaluation of unified memory technology on nvidia gpus," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, ser. CCGRID '15. IEEE Press, 2015, p. 10921098. [Online]. Available: https://doi.org/10.1109/CCGrid.2015.105

[15] J. Zhao and Y. Xie, "Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '12, 2012, pp. 81–87.

[16] M. Saxena and M. M. Swift, "Flashvm: Virtual memory management on flash." in *USENIX Annual Technical Conference*, 2010.

[17] A. Badam and V. S. Pai, "Ssdalloc: hybrid ssd/ram memory management made easy," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*. USENIX Association, 2011, pp. 211–224.

[18] "NVLink, Pascal and Stacked Memory: Feeding the Appetite for Big Data," NVIDIA Developer Blog, Tech. Rep., Mar. 2014. [Online]. Available: https://developer.nvidia.com/blog/nvlink-pascal-stacked-memory-feeding-appetite-big-data/

[19] *NAND Flash Memory MT29F64G08AB[C/E]BB, MT29F128G08AE[C/E]BB,MT29F256G08AK[C/E]BB*, Micron Technology, 2013.

[20] *NAND Flash Memory MT29F64G08CBAA[A/B], MT29F128G08C[E/F]AAA, MT29F128G08CFAAB*, Micron Technology, 2009.

[21] J. Nickolls and W. J. Dally, "The gpu computing era," *IEEE micro*, vol. 30, no. 2, pp. 56–69, 2010.

[22] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving gpu performance via large warps and two-level warp scheduling," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 308–317.

[23] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.