GPU Memory Management Solution Supporting Incomplete Pages

Li Shen(✉), Shiqing Zhang, Yaohua Yang, and Zhiying Wang

National University of Defense Technology, Changsha, China
zhangshiqing12@nudt.edu.cn

Abstract. Despite the increasing investment in integrated GPUs and next-generation interconnect research, discrete GPUs connected by PCI Express still account for the dominant position of the market, the management of data communication between CPU and GPU continues to evolve. This paper analyze the address translation overhead and migration latency introduced by this paged memory management solution in CPU-GPU heterogeneous systems. Based on the analysis, a new memory management scheme is proposed: paged memory management solution supporting incomplete pages, which can limit both address translation overhead and migration delay. "Incomplete" refers to a page that has only been partially migrated. This new memory management solution modifies the address translation and data migration process with only minor changes in hardware.

1 Introduction

The current GPU paged memory management solution is designed and implemented based on the unified memory [1–3]. When the requested page is missing on the device side, the system transfers the page to the local memory automatically. Paged memory management solution introduces two major overheads: address translation overhead and migration latency [4]. Due to the large GPU memory capacity and limited number of TLB entries, large pages can reduce address translation overhead. On the other hand, the migration delay is positively related to the page size.

In order to limit the address translation overhead and migration delay at the same time, this paper proposes a new GPU memory management scheme: paged memory management solution supporting incomplete pages. "Incomplete" refers to a page that has only been partially migrated. We implemented it on the gpgpu-sim simulator. Experimental results show that, compared to page memory management, it can reduce address translation overhead and migration latency at the same time.

Published by Springer Nature Switzerland AG 2018
F. Zhang et al. (Eds.): NPC 2018, LNCS 11276, pp. 174–178, 2018.
https://doi.org/10.1007/978-3-030-05677-3_20

This paper has the following three contributions:

1. We analyzed the address translation overhead and migration latency introduced by paged memory management solution. Based on this, a new memory management scheme is proposed: paged memory management solution supporting incomplete pages, which can limit both address translation overhead and migration delay.
2. We defined the "incomplete" page status, added records of the migrated range in the TLB and page table entries, modified the address translation operation, and divided it into two steps: check hit/miss and check whether it has been migrated, to support our new memory management scheme.
3. We modified the page migration operation and adjusted the functionality of GPU memory management unit (GMMU), so that it can specifies migration scope and merge requests when generating migration requests, to support our new memory management scheme and increase bandwidth utilization.

2 Related Work

Lustig and Martonosi [5] designed a fine-grained data dependency tracking mechanism to reduce migration delays. However, the system does not migrate data automatically, and introduces the overhead of tracking full/empty bits. Zheng et al. used the idle bandwidth to pre-migrate unrequested page based on the observation that PCI-E bandwidth utilization is low [4]. Agarwal et al. [6] use memory system information about the characteristics of heterogeneous memory systems to set the conditions for page migration. However, subsequent experiments have shown that the overhead exceeds the performance gains compared to the simple "migrate at the first request" strategy. Vesely et al. [7] analyzed address translation in heterogeneous systems and found that the cost in the GPU was an order of magnitude higher than the CPU. Ausavarungnirun et al. [8] designed and implemented Mosaic to provide application transparency support for multiple page sizes. In Mosaic, TLB and page tables need to support both large pages and small pages. The complex design introduces a lot of additional hardware modifications and overhead.

3 Paged Memory Management Solution Supporting Incomplete Pages

3.1 Overall Design

Compared with paged memory management solution, this new scheme has the following two differences. In the address translation process, in addition to determining whether the request is hit or miss, it is also checked whether the data in the requested address range has been migrated to the GPU memory. During the migration process, the system does not transfer the entire page. The generated migration request needs to specify the scope of the transfer. The architectural view of GPU MMU and TLBs in paged memory management solution supporting incomplete pages is shown in Fig. 1.

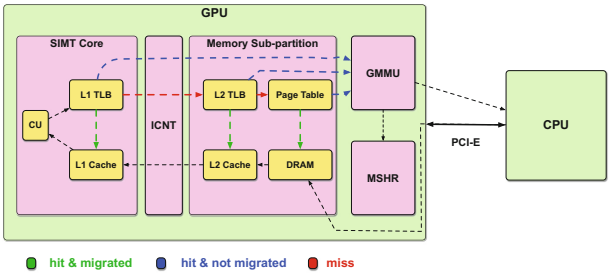

Fig. 1. Architectural view of GPU MMU and TLBs in paged memory management solution supporting incomplete pages

3.2 Address Translation

When querying each level of the TLB or the page table, the first step is to check whether the page is recorded. If it is missing, the request is passed to the next level TLB or page table. If it is hit, then check if the request address range has been migrated to the GPU. If it has been migrated, the address is translated and returned for cache access; if not, the GMMU informs the corresponding L1 TLB to suspend the request processing, generates a migration request and sends it to the CPU. When determining whether the request address has been migrated to the GPU, the page status and the migrated range are queried sequentially.

3.3 Migration Process

Since only the partial pages corresponding to the request are migrated, the migration request needs to inform the migration scope. It improves the ratio of calculation and memory access, reduces the unnecessary data transmission overhead, and significantly reduces the migration delay. In order not to waste CPU-GPU bandwidth, the scope of the migration request sets the minimum length based on the bandwidth value. The GMMU views the migration request waiting to be processed when generating a new migration request, and merges the requests whose migration range is less than the default threshold.

3.4 Data Access

When the requested data is migrated to the GPU local memory, the page table and the TLB are updated, then the request address is re-queried from the L1 TLB, and the cached and dram are accessed step by step with the converted address until the required data is obtained. Since the address translation and migration phases have already handled possible data misses, data can be obtained locally on the GPU.

Table 1. Simulator configuration

Simulator	GPGPU-Sim 3.x
GPU Arch	NVIDIA GTX-480 Fermi-like, 15 CUs @ 1.4 GHz
Caches	16 KB/CU L1, Mem Side 128 kB/Channel L2
TLBs	128-entry Per CU L1, 512-entry Shared L2
Clock Freqs	Core:IC:L2:DRAM 700:700:700:1024 (MHz)
GPU GDDR5	12-channels, 384 GB/sec aggregate
MSHR	128Entries/Memory Partition

4 Evaluation

4.1 Simulator and Benchmarks

We implemented our solution on gpgpu-sim [9,10]. The system configuration we use is shown in Table 1, including the key parameters of the GPU core and memory partition. It is assumed that the GPU memory is large enough so that no over-subscription will occur. We test under both 16 GB/sec and 32 GB/sec as representative of the current and future bandwidth [11]. The 10 benchmarks tested in our experimental are from ispass2009-benchmarks in gpgpu-sim. They come from different benchmark suites and are applied in various fields. BFS, MUM and NN are included in Rodinia, which is a general benchmark suite in GPGPU research.

4.2 Performance Comparison

We use 1 KB migration unit size as an example of multiple valid ranges migration, and the page size is 2 MB. Figure 2 shows that, when the bandwidth is 16 GB/sec, the performance of our new solution ("Incomplete") is much better than paged memory management solution ("Complete", which is called baseline in the following part). On average, our scheme improves from baseline's 82.43× deceleration to 1.36× acceleration compared with programmers controlled transfer.

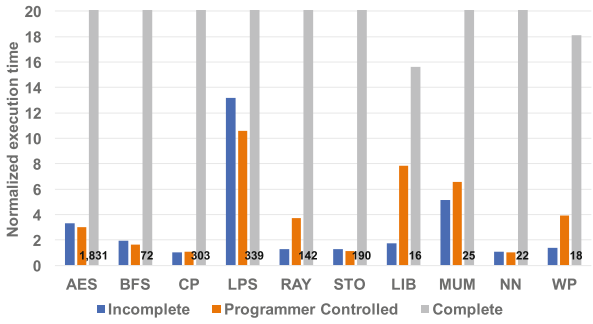

Fig. 2. Performance comparison under 16 GB/sec bandwidth. Workload execution time (lower is better) is normalized to ideal copy + execute overlap execution time.

5 Conclusion and Future Work

We defined the "incomplete" page status, added records of the migrated range in the TLB and page table entries, and divided the address translation operation into two steps: check hit/miss and check whether it has been migrated, to support our new memory management scheme. We modified the page migration operation and adjusted the functionality of GMMU, to support our new memory management scheme and increase bandwidth utilization.

But our experimental part is not perfect enough. There are many aspects to be tested and analyzed, including the performance comparison with Mosaic. We will complete the follow-up experiments in the next period of time, and analyze the experimental results to further improve our scheme. In addition, our scheme wastes part of memory capacity while reducing address translation overhead and migration delay, and this part of the cost needs to be further solved.

References

1. Harris, M.: Unified memory in CUDA 6. GTC On-Demand, NVIDIA (2013)
2. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: a unified graphics and computing architecture. Proc. IEEE Micro **28**(2), 39–55 (2008)
3. Landaverde, R., Zhang, T., Coskun, A.K., Herbordt, M.: An investigation of unified memory access performance in CUDA. In: Proceedings of IEEE High Performance Extreme Computing Conference, pp. 1–6 (2014)
4. Zheng, T., Nellans, D., Zulfiqar, A., Stephenson, M., Keckler, S.W.: Towards high performance paged memory for GPUs. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, pp. 345–357 (2016)
5. Lustig, D., Martonosi, M.: Reducing GPU offload latency via fine-grained CPU-GPU synchronization. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, pp. 354–365 (2013)
6. Agarwal, N., Nellans, D., Stephenson, M., O'Connor, M., Keckler, S.W.: Page placement strategies for GPUs within heterogeneous memory systems. ACM SIGPLAN Not. **50**, 607–618 (2015)
7. Vesely, J., Basu, A., Oskin, M., Loh, G.H., Bhattacharjee, A.: Observations and opportunities in architecting shared virtual memory for heterogeneous systems. In: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, pp. 161–171 (2016)
8. Ausavarungnirun, R., et al.: Mosaic: a GPU memory manager with application-transparent support for multiple page sizes. Carnegie Mellon University, SAFARI Research Group, Technical report TR-2017-003 (2017)
9. Bakhoda, A., Yuan, G.L., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, pp. 163–174 (2009)
10. Aamodt, T.M., et al.: GPGPU-Sim 3.x manual (2012)
11. Ajanovic, J.: PCI express 3.0 overview. In: Proceedings of Hot Chips: A Symposium on High Performance Chips (2009)