EasyChair Preprint

№ 4179

Dynamic GMMU Bypass for Address Translation in Multi-GPU Systems

Jinhui Wei, Jianzhuang Lu, Qi Yu, Chen Li and Yunping Zhao

September 13, 2020

Dynamic GMMU Bypass for Address Translation in Multi-GPU Systems

Jinhui Wei[1], Jianzhuang Lu[1], Qi Yu[1], Chen Li[1(✉)], and Yunping Zhao[1]

College of Computer Science, National University of Defense Technology, Changsha 410073, China
{jinhui_wei,lichen, zhaoyunping}@nudt.edu.cn
{lujz1977}@163.com
{fishflag}@126.com

Abstract. The ever increasing application footprint raises challenges for GPUs. As Moore's Law reaches its limit, it is not easy to improve single GPU performance any further; instead, multi-GPU systems have been shown to be a promising solution due to its GPU-level parallelism. Besides, memory virtualization in recent GPUs simplifies multi-GPU programming. Memory virtualization requires support for address translation, and the overhead of address translation has an important impact on the system's performance. Currently, there are two common address translation architectures in multi-GPU systems, including distributed and centralized address translation architectures. We find that both architectures suffer from performance loss in certain cases. To address this issue, we propose GMMU Bypass, a technique that allows address translation requests to dynamically bypass GMMU in order to reduce translation overhead. Simulation results show that our technique outperforms distributed address translation architecture by 6% and centralized address translation architecture by 106% on average.

Keywords: Multi-GPU system· Memory virtualization· Address translation architecture

1 Introduction

Graphics Processing Units (GPUs) have been widely used in graph analytics [16,17], large scale simulation [6,14], and machine learning [8,13] due to its massive thread-level parallelism. Over the years, with the development of big data, the application footprint has increased rapidly, which raises challenges for GPUs. What is worse, as Moore's Law reaches its limit [1], improving GPU performance through integrating more transistors on a single die is more difficult than ever before. Instead, multi-GPU systems [4,7,18] have been shown to be a promising solution due to its GPU-level parallelism. Nowadays, multi-GPU systems have been used in data centers to improve the performance of cloud computing [9].

Recent support for memory virtualization [5,19,20] in GPUs has simplified programming and improved programming productivity. Memory virtualization requires the support of address translation. Also the details of memory hierarchy from mainstream GPU manufacturers, such as NVIDIA, AMD, and Intel, have not been published, it is accepted that current GPU supports TLB-based address translation [2,4]. Recent research [2,3,11,12] has shown that the efficiency of address translation has an important impact on GPU performance.

Currently, there are two common address translation architectures in multi-GPU systems, namely, centralized address translation architecture ("centralized architecture" for short) and distributed address translation architecture ("distributed architecture" for short). The major difference between these two architectures is that distributed architecture uses a GMMU (GPU Memory Management Unit) in each GPU node (in the system) to manage address translation for that GPU. When an address translation misses L2 TLB, the request is sent to GMMU first for page table walk in distributed architecture; while, in centralized architecture, the request is directly sent to IOMMU (Input Output Memory Management Unit) on the CPU side for translation.

In this paper, we make an in-depth analysis of these two architectures. In terms of hardware overhead, centralized architecture causes less overhead due to the absence of GMMU. In terms of performance, distributed architecture outperforms centralized architecture on average, because GMMU reduces the frequency of remote translation (translation requests are sent to IOMMU) for local translation requests (translation requests find mappings in local memory). However, we find that distributed architecture suffers from performance slowdown in certain cases. For example, for those access requests that demand for shared data (residents in other GPU node), the address translation requests are sent to page table walker in GMMU if they miss L2 TLB. As the shared data does not resident in the local GPU memory, the translation requests cannot find address mappings in local memory either. These requests are then sent to IOMMU for further translation. These unnecessary page table walks consume additional power and incur performance degradation.

To address this issue, we propose GMMU Bypass in distributed architecture to reduce unnecessary page table walks and improve address translation performance. GMMU Bypass uses two fixed thresholds to predict according to the variance in access behavior. Simulation results show that GMMU Bypass is effective at reducing the overhead of handling address translation requests.

This paper makes the following major contributions:

- To our knowledge, this is the first work to provide in-depth analysis of two address translation architectures from hardware overhead and performance points of view in multi-GPU system.
- We propose GMMU Bypass, a technique that bypasses GMMU selectively to improve the performance of multi-GPU system by profiling and predicting the memory access behavior. We evaluate the performance of our design and results show that GMMU Bypass outperforms the distributed architecture by 6% and centralized architecture by 106% averagely.

2 Background

In this section, we introduce the background on multi-GPU systems, including programming models, remote data access mechanisms, and address translation architectures.

2.1 Programming Models

Currently, there exist two programming models in multi-GPU system: discrete model and unified model [15]. Discrete model dispatches kernels to each GPU node for execution, as a result, programmers have to rewrite the code developed for single GPU in order to make it executable on multi-GPU systems. While, unified model dispatches tasks at CTA (Cooperative Thread Array) granularity, which means the CTAs of a single kernel can be dispatched to different GPU nodes for execution. In this case, the code developed for single GPU can run seamlessly on multi-GPU systems without any modification. Due to the programming convenience, the research community focuses on unified model. This paper also targets unified model.

2.2 Remote Data Access Mechanisms

There exist three remote data access mechanisms in multi-GPU system: direct cache access, page migration, and first touch migration. *Direct cache access* means that a GPU directly accesses the L2 cache of a remote GPU node to retrieve the requested data through RDMA (Remote DMA) [4,10,18]. In this case, the corresponding page will not be migrated to the requested GPU. *Page migration* refers to migrating a page from the GPU which it residents in to the requested GPU in case of a page fault. *First touch migration* is a special case of page migration. It means that CPU migrates the page to the GPU that demands it first, which is used for data allocation generally.

2.3 Address Translation Architectures

There exist two address translation architectures in multi-GPU systems: centralized/distributed architecture. The major difference between the two architectures is whether the GPU node has a GMMU. The address translation process in these two architectures is shown as follows.

Centralized Architecture The address translation process in centralized architecture is shown in Fig. 1. The request first accesses the L1 TLB to check for the address mapping. On an L1 TLB miss, the request accesses L2 TLB (①). If the request also misses L2 TLB, the request is sent to IOMMU (on the CPU side) for further translation (②). IOMMU performs page table walk and sends the required address mapping to the GPU (③). The GPU completes the data access using the translated address. If the data residents in local memory, the data is retrieved from its memory hierarchy; otherwise, the data is retrieved from remote GPU through RDMA (④).

Distributed Architecture The address translation process in distributed architecture is shown in Fig. 2. The request accesses L1 TLB first for address translation. On an L1 TLB miss, the request accesses L2 TLB (①). If it also misses L2 TLB, the request is sent to page table walker in GMMU for page walk (②). If the request finds the desired page table entry during page walk, the address translation is finished; otherwise, the request is sent to IOMMU for further translation (③). IOMMU performs page table walk and sends back the required address translation mapping (④). The GPU retrieves the data from its local memory or remote GPU via RDMA (⑤).

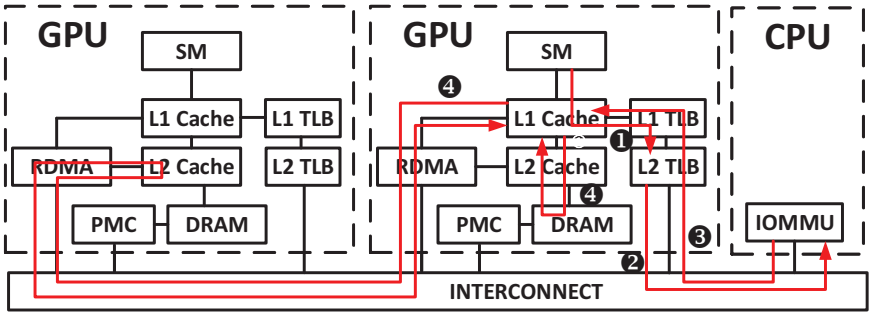


Fig. 1: Centralized address translation architecture

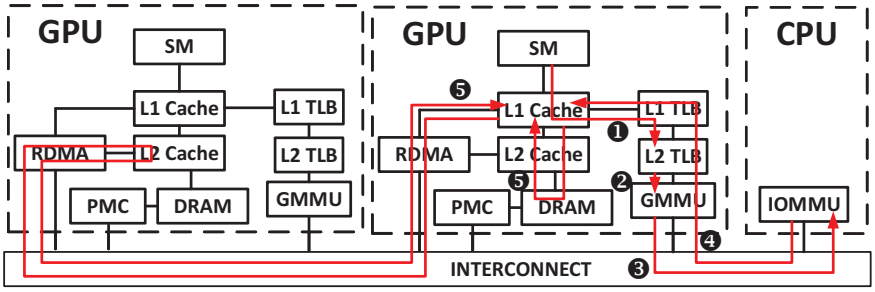


Fig. 2: Distributed address translation architecture

3 Motivation

The major difference between centralized architecture and distributed architecture is whether address translation requests are sent to GMMU for page table walk. For centralized architecture, when translation requests miss L2 TLB, the requests are directly sent to IOMMU for further translation; while, for distributed architecture, the requests are sent to GMMU for page table walk.

Despite the fact that distributed architecture incurs higher hardware overhead due to GMMU, it usually outperforms centralized architecture in terms

of performance. Through further analysis, we find that the address translation requests that miss L2 TLB touch either a local page (residents in local memory) or a remote page (residents in a remote GPU node). If it touches a local page, for distributed architecture, the address translation completes after page table walk, and thus, there is no need to access the IOMMU. However, for centralized architecture, no matter where the page residents, the request is always sent to IOMMU for translation. As the communication between CPU and GPU via PCIe incurs significant overhead (refers to latency), the remote translation (accesses IOMMU) causes much longer latency than page table walk. Therefore, for address translation requests that touch local pages, distributed architecture can reduce translation overhead and improves performance.

However, if the address translation request touches a remote page, distributed architecture may cause slight performance slowdown. This is because GMMU does not store the page table entry of remote pages, therefore, the address translation request cannot find the desired address mapping in local memory after page table walk. In other words, the page table walk is unnecessary for these requests. These unnecessary page table walks waste power and may cause performance degradation.

To quantitatively show the discrepancy of two architectures, we evaluate the performance of these two architectures and an ideal scheme, which is shown in Fig. 3. The ideal scheme can predict the exact destination for each translation request, and thus, it achieves the best performance. The experimental methodology can be found in Section 5. We have two observations from Fig. 3. First, we can see that though distributed architecture significantly outperforms centralized architecture for `MT`, `FFT`, `KM`, and `ST`, it does worse than centralized architecture for `RL`, `FIR`, and `MP`. This result corroborates our analysis that unnecessary page table walks may harm the performance of distributed architecture. Second, we discover that the performance of the ideal scheme is better than distributed architecture, which means that there are a great number of unnecessary page table walks existing in distributed architecture. So we can propose a mechanism that selects better destinations to reduce unnecessary page table walks for improving performance.

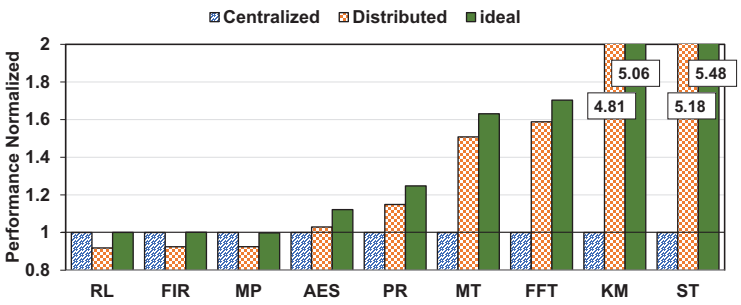


Fig. 3: Comparison of different address translation architecture

4 GMMU Bypass

To reduce unnecessary page table walks and maintain the advantages of GMMU for distributed architecture, we propose GMMU Bypass, a GMMU-side mechanism that 1) allows address translation requests to directly access IOMMU without walking the GPU page table, or 2) simultaneously sends translation requests to GMMU and IOMMU. Its architecture is shown in Fig. 4.

GMMU Bypass is a simple mechanism, it counts the number of address translation requests that touch local pages and remote pages, and calculates the ratio periodically (L/T ratio in the figure, L is the number of local pages, T is the number of all pages). If the ratio is larger than a threshold (*T1*), which means more requests touch local pages in the current epoch, then the control logic predicts more requests touching local pages in the next epoch and disables GMMU bypass. If the ratio is smaller than a threshold (*T2*), which means more requests touche remote pages in the current epoch, then it predicts that this is also the case in the next epoch and enables GMMU bypass.

However, we find that when the ratio locates between *T1* and *T2*, the prediction accuracy is low as a result of random accesses to local pages and remote pages. The random accesses do not show a domination trend and accesses to local/remote pages account for a certain percentage. In this case, it is not appropriate to send all requests to GMMU only or IOMMU only. Instead, our mechanism sends all requests to GMMU and IOMMU simultaneously. If the request can find the desired address mapping after page table walk, then the response from IOMMU will be discarded; while, if the request cannot find the address translation, it will be discarded, since there has been a request sent to IOMMU for further translation.

By simultaneously sending requests to GMMU and IOMMU, the requests can be handled without unnecessary page table walks. Therefore, the latency caused by awaiting page table walks for remote translation in distributed architecture is eliminated. However, the performance gain comes at a cost. Sending requests to both MMUs simultaneously generates unnecessary page table walks in GMMU or IOMMU, which may cause congestion in GMMU and IOMMU. According to our evaluation, setting proper thresholds (empirically set T1 to 0.85 and T2 to 0.15) can both achieve high performance and limit the congestion effectively.

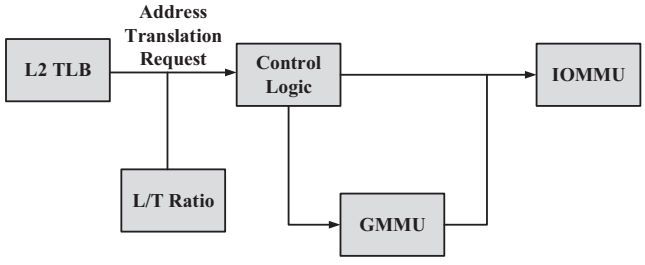


Fig. 4: GMMU Bypass

5 Methodology

We evaluate Bypass GMMU with MGPUSim [15], a multi-GPU simulator that supports multi-GPU system simulation. This simulator has been validated against AMD multi-GPU systems.

5.1 Experimental Setup

We evaluate a multi-GPU system with 4 AMD GPUs. The configuration is shown in Table 1. Each GPU consists of 4 SEs (Shader Engine) and each SE consists of 9 CUs (Compute Unit). Therefore, each GPU has 36 CUs. Each CU has a private L1 vector TLB and all CUs in a GPU share an L2 TLB. Each Shader Engine is equipped with an L1 instruction TLB and an L1 scalar TLB. The 4 GPUs in the system are connected via PCIe-v3 link with bandwidth of 16GB/s in each direction. CPU and 4 GPUs are also connected via the PCIe-v3 link. The IOMMU on CPU side supports 8 concurrent page table walks and the GMMU in GPU supports 64 concurrent page table walks. The page size is 4KB, which is the default size in current GPUs.

5.2 Workloads

We use workloads from AMD APPSDK, Hetero-Mark, DNN, and SHOC benchmark suites for evaluation. These workloads cover a wide range of domains, including machine learning, graph analytics, numerical computation, etc. The average dataset size of these workloads is 64 MB. Long simulation time prevents us from evaluating workloads with larger footprint. Selected workloads are shown in Table 2.

6 Evaluation

We first compare the performance of GMMU Bypass with centralized architecture, distributed architecture, and an ideal scheme. We also make an in-depth analysis behind the performance result. Finally, we estimate the overhead of our mechanism.

Table 1: Multi-GPU System Configuration

Component	Configuration	Number per GPU
CU	1.0GHz	36
DRAM	512MB HBM	8
L1 TLB	1 set, 32-ways	44
L2 TLB	8 set, 32-ways	1
IOMMU	8 Page Table Walkers	-
Intra-GPU Network	Single-stage XBar	1
Intra-Device Network	16GB/s PCIe-v3	-
GMMU	Page Table Walkers	64

Table 2: Workloads Characteristics

Abbv.	Application	Benchmark Suite	Mem.Footprint
RL	Relu	DNN	64M
FIR	Finite Impulse Resp.	Hetero-Mark	64M
MP	Maxpooling	DNN	64M
AES	AES-256 Encryption	Hetero-Mark	64M
PR	PageRank Algorithm	Hetero-Mark	64M
MT	Matrix Transpose	AMDAPPSDK	64M
FFT	Fast Fourier Transform	SHOC	64M
KM	KMeans Clustering	Hetero-Mark	64M
ST	Stencil 2D	SHOC	64M

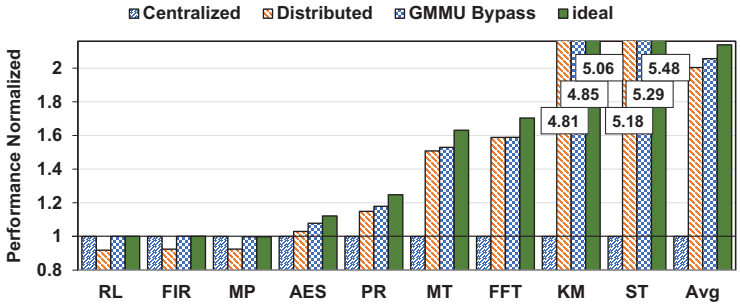


Fig. 5: Performance normalized to each address translation architecture

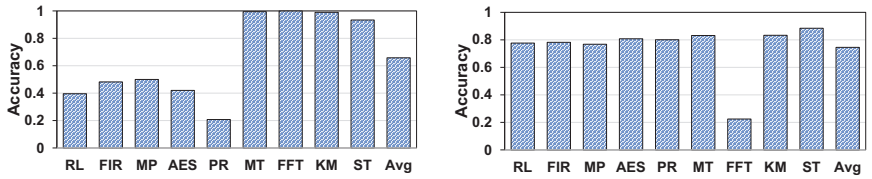
6.1 Performance

GMMU Bypass exposes several design parameters, including *sampling epoch*, *T1* and *T2*, We set sampling epoch to 10µs, T1 to 0.85 and T2 to 0.15 as we empirically find that these values yields best trade-off between performance and overhead. The performance comparison is shown in Fig. 5. We make three observations. First, our scheme outperforms distributed architecture for `RL`, `FIR`, and `MP`, which distributed architecture suffers. Second, our scheme achieves similar or slightly higher performance with distributed architecture for the rest of applications, which distributed architecture performs well. Third, the ideal scheme performs best for all applications. On average, our scheme achieves 6% performance improvement over distributed architecture and is within 96% of the ideal scheme.

6.2 Analysis

We provide an in-depth analysis of our proposed technique.

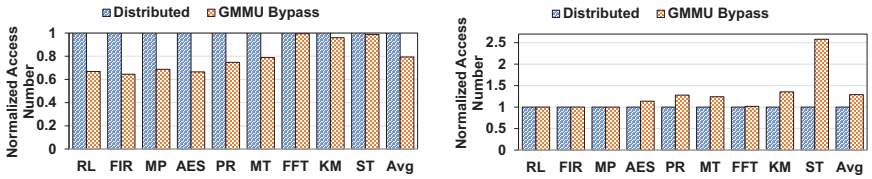
Prediction Accuracy When the L/T ratio is larger than *T1* or smaller than *T2*, our technique disables GMMU bypass or enables GMMU bypass correspondingly. Although in this case, L/T ratio shows a domination trend, the prediction



(a) Result when L/T ratio is larger than 0.85

(b) Result when L/T ratio is smaller than 0.15

Fig. 6: Prediction accuracy of our technique when L/T ratio is larger than 0.85 and smaller than 0.15



(a) Result in GMMU

(b) Result in IOMMU

Fig. 7: The number of reduplicate page table walks in GMMU and IOMMU

is not always accurate. We check this by calculating the prediction accuracy of our scheme. The result is shown in Fig. 6.

It shows that when L/T ratio is larger than 0.85 (*T1*), our technique achieves high prediction accuracy for `MT`, `FFT`, `KM`, and `ST`, but achieves relative low accuracy for `RL`, `FIR`, `MP`, `AES`, and `PR`. Through further analysis, we find that for these latter five applications, our technique only collects a few translation requests in the sampling epochs. The limited number of translation requests does not provide enough information for prediction, thereby lowers prediction accuracy. When L/T ratio is smaller than 0.15 (*T2*), our technique achieves relative high prediction accuracy except `FFT`. We find that `FFT` also suffers from limited number of translation requests in sampling epochs. The average prediction accuracy is 66% and 75% in these two cases, respectively.

Reduplicate page table walks As it is not easy to predict the accurate access behavior (translation requests access local pages or remote pages)) when L/T ratio lies between 0.15 and 0.85, our techniques chooses to send the request to GMMU and IOMMU simultaneously. This generates reduplicate page table walks in GMMU and IOMMU since there is only one effective page table walk. To quantitatively show the number of reduplicate page table walks, we record the number of page table walks in GMMU and IOMMU respectively. The result is shown in Fig. 7.

Although our technique increases the number of duplicate page table walks in GMMU, the total number of page table walks is still smaller than that in distributed architecture. This is because by enabling GMMU bypass, our technique can reduce unnecessary page table walks. In IOMMU, our technique does not increase the number of page table walks dramatically except `ST`. This is because `ST` sends more translation requests when L/T ratio lies between 0.15 and 0.85, thereby increases the number of duplicate page table walks in IOMMU significantly. The number of all page table walks increases 14% and the performance improves 11% in `ST` specificly. On average, our technique reduces the number of page table walks by 21% in GMMU, and increases the number of page table walks by 29% in IOMMU compared to distributed architecture. Therefore, our technique does not incur a significant number of reduplicate page table walks in both GMMU and IOMMU.

6.3 Hardware Cost

Our technique is simple and incurs negligible hardware costs. It only needs two 8-bit counters to record the number of translation requests that touch local pages and remote pages respectively. In addition, a simple ALU is enough for calculating the L/T ratio.

7 Related Work

As far as we know, this paper is the first to optimize the address translation workflow for multi-GPU systems. In this section, we introduce previous research focusing on address translation designs on GPUs and performance optimization for multi-GPU systems.

7.1 Address Translation on GPU

Since the introduction of unified memory, there have been several works that target address translation designs on GPUs. J. Power *et al.* and B. Pichai *et al.* were among the first to explore such designs. In J. Power's design [12], per-CU private L1 TLB, a highly-threaded page table walker and page walk cache are essential components for efficient address translation on GPUs. B. Pichai's design includes per-CU private TLB and page table walker [11]. The authors showed the importance of making the warp scheduler to be aware of TLB design. R. Ausavarungnirun *et al.* showed that replacing the page walk cache with a shared L2 TLB can improve the performance of address translation [2].

7.2 Performance Optimization for Multi-GPU Systems

Despite multi-GPU systems utilize GPU-level parallelism, it suffers from inefficiency in certain cases. The research community proposes several optimization techniques to improve multi-GPU performance [4,7,18]. T. Baruah *et al.* proposed Griffin [4], a page allocation strategy to reduce the impact of bandwidth

by making more remote accesses to be local accesses. V.Young *et al.* proposed CARVE [18], a hardware mechanism that stores recently accessed remote shared data in a dedicated region of GPU memory. G. Kim *et al.* proposes a strategy to allocate CTAs in multi-GPU system [7]. This strategy can improve the spatial locality of data access and improves the address translation efficiency for distributed architecture. Our work focuses on the address translation in distributed architecture.

8 Conclusion

The address translation efficiency has an important impact on the performance of multi-GPU systems. Although distributed architecture significantly outperforms centralized architecture for a majority of workloads, it suffers in certain cases due to unnecessary page table walks. In this paper, we propose GMMU Bypass, a technique aims at reducing the overhead of unnecessary page table walks in GMMU. The simulation result shows that our technique achieves 6% performance improvement over distributed architecture and is within 96% of an ideal scheme, which shows the effectiveness of our technique.

Acknowledgement. This work is partially supported by Research Project of NUDT ZK20-04, PDL Foundation 6142110180102, Science and Technology Innovation Project of Hunan Province 2018XK2102 and Advanced Research Program 31513010602-1.

References

1. Arunkumar, A., Bolotin, E., Cho, B., Milic, U., Ebrahimi, E., Villa, O., Jaleel, A., Wu, C.J., Nellans, D.: Mcm-gpu: Multi-chip-module gpus for continued performance scalability. ACM SIGARCH Computer Architecture News **45**(2), 320–332 (2017)
2. Ausavarungnirun, R., Landgraf, J., Miller, V., Ghose, S., Gandhi, J., Rossbach, C.J., Mutlu, O.: Mosaic: a gpu memory manager with application-transparent support for multiple page sizes. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 136–150 (2017)
3. Ausavarungnirun, R., Miller, V., Landgraf, J., Ghose, S., Gandhi, J., Jog, A., Rossbach, C.J., Mutlu, O.: Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency. ACM SIGPLAN Notices **53**(2), 503–518 (2018)
4. Baruah, T., Sun, Y., Dinçer, A.T., Mojumder, S.A., Abellán, J.L., Ukidave, Y., Joshi, A., Rubin, N., Kim, J., Kaeli, D.: Griffin: Hardware-software support for efficient page migration in multi-gpu systems. In: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 596–609 (Feb 2020). https://doi.org/10.1109/HPCA47549.2020.00055
5. Ganguly, D., Zhang, Z., Yang, J., Melhem, R.: Interplay between hardware prefetcher and page eviction policy in cpu-gpu unified virtual memory. In: Proceedings of the 46th International Symposium on Computer Architecture. pp. 224–235 (2019)

6. Jermain, C., Rowlands, G., Buhrman, R., Ralph, D.: Gpu-accelerated micromagnetic simulations using cloud computing. Journal of Magnetism and Magnetic Materials **401**, 320–322 (2016)
7. Kim, G., Lee, M., Jeong, J., Kim, J.: Multi-gpu system design with memory networks. In: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 484–495. IEEE (2014)
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
9. Li, C., Sun, Y., Jin, L., Xu, L., Cao, Z., Fan, P., Kaeli, D.R., Ma, S., Guo, Y., Yang, J.: Priority-based pcie scheduling for multi-tenant multi-gpu system. IEEE Computer Architecture Letters (2019)
10. NVIDIA, T.: V100 gpu architecture, 2017. Whitepaper, online at nvidia. com, accessed **9** (2019)
11. Pichai, B., Hsu, L., Bhattacharjee, A.: Architectural support for address translation on gpus: Designing memory management units for cpu/gpus with unified address spaces. ACM SIGARCH Computer Architecture News **42**(1), 743–758 (2014)
12. Power, J., Hill, M.D., Wood, D.A.: Supporting x86-64 address translation for 100s of gpu lanes. In: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA). pp. 568–578. IEEE (2014)
13. Raina, R., Madhavan, A., Ng, A.Y.: Large-scale deep unsupervised learning using graphics processors. In: Proceedings of the 26th annual international conference on machine learning. pp. 873–880 (2009)
14. Sanaullah, A., Mojumder, S.A., Lewis, K.M., Herbordt, M.C.: Gpu-accelerated charge mapping. In: 2016 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–7. IEEE (2016)
15. Sun, Y., Baruah, T., Mojumder, S.A., Dong, S., Gong, X., Treadway, S., Bao, Y., Hance, S., McCardwell, C., Zhao, V., et al.: Mgpusim: Enabling multi-gpu performance modeling and optimization. In: Proceedings of the 46th International Symposium on Computer Architecture. pp. 197–209 (2019)
16. Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A., Owens, J.D.: Gunrock: A high-performance graph processing library on the gpu. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 1–12 (2016)
17. Wu, Y., Wang, Y., Pan, Y., Yang, C., Owens, J.D.: Performance characterization of high-level programming models for gpu graph analytics. In: 2015 IEEE International Symposium on Workload Characterization. pp. 66–75. IEEE (2015)
18. Young, V., Jaleel, A., Bolotin, E., Ebrahimi, E., Nellans, D., Villa, O.: Combining hw/sw mechanisms to improve numa performance of multi-gpu systems. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 339–351. IEEE (2018)
19. Zheng, T., Nellans, D., Zulfiqar, A., Stephenson, M., Keckler, S.W.: Towards high performance paged memory for gpus. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 345–357. IEEE (2016)
20. Ziabari, A.K., Sun, Y., Ma, Y., Schaa, D., Abellán, J.L., Ubal, R., Kim, J., Joshi, A., Kaeli, D.: Umh: A hardware-based unified memory hierarchy for systems with multiple discrete gpus. ACM Transactions on Architecture and Code Optimization (TACO) **13**(4), 1–25 (2016)