Priority-Based PCIe Scheduling for Multi-Tenant Multi-GPU Systems

Chen Li, Yifan Sun, Lingling Jin, Lingjie Xu, Zheng Cao, Pengfei Fan, David Kaeli, Sheng Ma, Yang Guo, and Jun Yang

Abstract—Multi-GPU systems are widely used in data centers to provide significant speedups to compute-intensive workloads such as deep neural network training. However, limited PCIe bandwidth between the CPU and multiple GPUs becomes a major performance bottleneck. We observe that relying on a traditional Round-Robin-based PCIe scheduling policy can result in severe bandwidth competition and stall the execution of multiple GPUs. In this article, we propose a priority-based scheduling policy which aims to overlap the data transfers and GPU execution for different applications to alleviate this bandwidth contention. We also propose a dynamic priority policy for semi-QoS management that can help applications to meet QoS requirements and improve overall multi-GPU system throughput. Experimental results show that the system throughput is improved by 7.6 percent on average using our priority-based PCIe scheduling scheme as compared with a Round-Robin-based PCIe scheduler. Leveraging semi-QoS management can help to meet defined QoS goals, while preserving application throughput.

Index Terms—Multi-GPU, multi-tenant, PCIe scheduling

1 Introduction

Recently, as GPUs have quickly become standard computing devices in datacenter systems, cloud computing vendors are starting to deploy multi-GPU systems on the cloud and deliver GPUs “as a service” [1], [2]. In cloud-based multi-GPU systems, a typical configuration is to allocate each user (tenant) with a certain number of dedicated GPUs, while virtualizing the CPU and the PCIe connection, so they can be shared by multiple users [3], [4], [5].

Typical GPU workloads follow the “*copy-then-execute*” model. The execution on the GPU side cannot start until data is fully copied from the CPU to the GPU. As shown in Fig. 1, the PCIe bandwidth between a CPU and a GPU remains constant moving a single GPU system to a multi-GPU architecture, leading to serious bandwidth competition among multiple GPUs when communicating with the CPU, which further delays the start of GPU execution.

By default, the communication traffic from the CPU to the attached GPUs is scheduled in a *Round-Robin* (RR) manner. RR scheduling attempts to guarantee fairness among all GPUs, but introduce the delay of key memory packets and impact the throughput of the corresponding GPU execution. Observed from production multi-GPU systems, this bandwidth competition causes severe performance degradation, especially for memory-bound workloads. Moreover, multi-tenant system users cannot manage their own data movement as each user is treated agnostically. Therefore, it is necessary to *schedule* the PCIe traffic associated with different GPUs to improve the overall system throughput.

- *C. Li, S. Ma, and Y. Guo are with the National University of Defense Technology, Changsha 410073, China. E-mail: {lichen, masheng, guoyang}@nudt.edu.cn.*
- *Y. Sun and D. Kaeli are with Northeastern University, Boston, MA 02115. E-mail: yifansun@coe.neu.edu, kaeli@ece.neu.edu.*
- *L. Jin, L. Xu, Z. Cao, and P. Fan are with Alibaba Inc., Hangzhou, Zhejiang 311121, China. E-mail: {l.jin, lingjie.xu, zhengzhi.cz, qiheng.fpf}@alibaba-inc.com.*
- *J. Yang is with the University of Pittsburgh, Pittsburgh, PA 15260. E-mail: juy9@pitt.edu.*

Manuscript received 21 Oct. 2019; accepted 13 Nov. 2019. Date of publication 22 Nov. 2019; date of current version 13 Jan. 2020.
(Corresponding author: Chen Li.)
Digital Object Identifier no. 10.1109/LCA.2019.2955119

In this paper, we exploit a priority-based PCIe scheduling policy and describe semi-QoS application management on CPU-GPU communication to improve multi-GPU throughput. Memory transfer commands with smaller data sizes are prioritized at runtime to achieve higher throughput. If a task has a specified QoS goal, but the goal is estimated that it will not be met, the task’s priority level is escalated to meet the requirement. Experimental results show that system throughput is improved by 7.6 percent on average with priority-based PCIe scheduling compared with the Round-Robin-based PCIe scheduling. The semi-QoS management can also meet defined QoS goals, achieving a 5.3 percent performance improvement as compared with RR PCIe scheduling.

2 Motivation

Current commercial multi-GPU systems consist of four to eight GPUs interconnected with a PCIe or NVLink fabric. Traditionally, four GPUs share a host CPU and a PCIe bus. Every two GPUs are connected to a PCIe switch which also connects to the root complex of the PCIe bus, as shown in Fig. 2. Constrained by the inter-GPU or CPU-GPU communication bandwidth, a multi-GPU system cannot scale performance linearly as the increasing number of GPUs. The low bandwidth and high latency associated with the current inter-GPU/CPU-GPU fabric can be a bottleneck for the system performance.

Fig. 3 shows the performance degradation due to the PCIe bandwidth contention between CPUs and GPUs. We run two deep neural network models *ResNet50* [6] and *DeepInterest* [6] and multiple instances of each (tasks), totaling eight tasks in all possible combinations. Each task executes either the training of *ResNet50* or *DeepInterest* on one GPU. Different tasks are bound to different CPU cores to avoid interference. We have two key observations from Fig. 3. First, since tasks are independent of each other, and we intentionally allocate them on different CPU cores and GPUs, so most of the slowdown will be due to contention of CPU-GPU PCIe bandwidth, resulting in performance degradation of both *ResNet50* and *DeepInterest* training, as each execution needs to wait for the data transfer. Second, workloads have different sensitivities to PCIe contention. *DeepInterest* is more sensitive, while *ResNet50* is not. The slowdown of *ResNet50* is around 5 percent over an isolated execution, even when seven instances are executing concurrently, while *DeepInterest* sees more than a 20 percent degradation with one instance competing with seven *ResNet50* instances. On average, we observe a 18.1 percent slowdown for *DeepInterest*. We conclude that the contention on the PCIe connection significantly degrades the performance of multi-tenant multi-GPU systems, especially for bandwidth-sensitive workloads. Delivering an improved PCIe scheduling policy is mandated to reduce such contention.

Existing PCIe connections in multi-GPU systems adopt a RR scheduling policy. For each data transfer request, the DMA engine breaks down data transfers into smaller packets and buffers them in the network interface. The PCIe arbiter selects a packet from a different application to transfer during each time slot. Assuming two applications have a similar amount of data to transfer to their GPU, an RR-based PCIe connection would complete the data transfer in approximately the same time, which is about twice the time of a single data transfer without contention. Consequently, the GPUs start executing kernels at approximately the same time, leaving their computing pipelines idle during the data transfer. Instead, if we only transfer the data from one application first, the GPU can start executing the kernel for this application earlier. As the GPU is executing the application, the PCIe connection can transfer the data for the other applications, overlapping transfers and

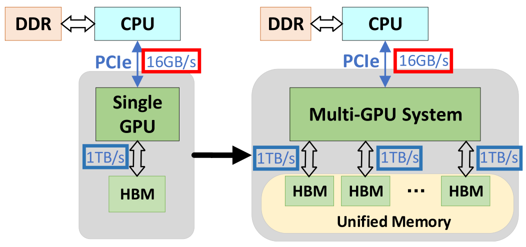

Fig. 1. PCIe bandwidth bottleneck in multi-GPU systems.

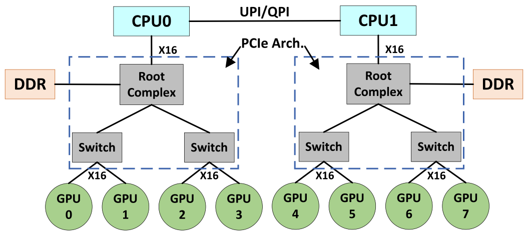

Fig. 2. Multi-GPU topology.

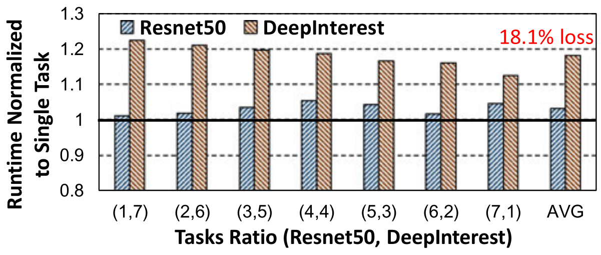

Fig. 3. Slowdown due to PCIe bandwidth contention. (Collected from a multi-GPU system with 8 NVIDIA Volta GPUs.)

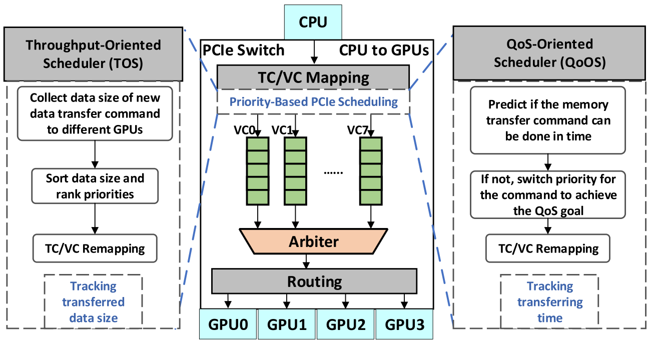
CPU
Throughput-Oriented Scheduler (TOS)
Collect data size of new data transfer command to different GPUs
Sort data size and rank priorities
TC/VC Remapping
Tracking transferred data size
PCIe Switch
CPU to GPUs
TC/VC Mapping
Priority-Based PCIe Scheduling
VC0
VC1
VC7
Arbiter
Routing
GPU0
GPU1
GPU2
GPU3
QoS-Oriented Scheduler (QoOS)
Predict if the memory transfer command can be done in time
If not, switch priority for the command to achieve the QoS goal
TC/VC Remapping
Tracking transferring time

Fig. 4. High-level overview of a PCIe switch (The process of a packet sending from the CPU side to GPU side).

execution. Ideally, this approach can effectively hide the data-transfer latency across multiple GPUs and improve resource utilization over the execution of multiple iterations.

Previous work has focused on software and hardware mechanisms to address Non-Uniform Memory Access (NUMA) bottlenecks on GPUs. Arunkumar et al. [7] proposed MCM-GPU, which integrates multiple GPU modules on a package to provide high bandwidth for inter-GPU traffic. Milic et al. [8] described links that are dynamically and adaptively reconfigurable, in order to optimize the inter-socket bandwidth. They also provide caching of remote data in L2 cache to reduce the effect of NUMA bottleneck. Young et al. [9] proposed CARVE that caches remote data in GPU memory to improve the NUMA performance of multi-GPU systems. All this prior work is focused on the bottleneck of inter-GPU communication for single applications. To our knowledge, this is the first paper to address the multi-tenant execution problem through better PCIe scheduling, as commonly seen in cloud infrastructures.

3 Design

3.1 Baseline Round-Robin Scheduling Design

The baseline multi-GPU system in this paper consists of four GPUs connected to one host CPU, which is among the most widely adopted architecture in commercial multi-GPU servers. The data transfer between CPU and GPU is managed by the DMA engine. A memory copy command from CPU to GPU can be divided into multiple data packets. Each packet on the PCIe link is labeled with the destination GPU (ID_D) and the packet type (ID_T). To transfer data from the CPU to GPUs via the DMA engine and PCIe switches, packets are first mapped to a particular virtual channel (VC) according to its traffic class (TC) as shown in Fig. 4. TC can be calculated according to

$$TC = N_T ID_D + ID_T, \tag{1}$$

where N_T is the total number of packet types. TCs are directly mapped to different VCs by

$$VC = TC \bmod N_{VC}. \tag{2}$$

The RR policy is set as the VC arbitration policy in the PCIe switch. All VCs have the same priority so that packets from different VCs are fetched and handled one-by-one in the RR order. Although the TC could be defined by users, we assign the TC for each memory transfer command using the driver, in order to support transparent multi-tenant management.

In this case, if there are packets in VCs that are queued, the arbiter forwards one following the VC arbitration policy to the routing logic in each cycle. The routing logic then directs packets to their destination GPUs. Since packets being sent to different GPUs are assigned different VCs, each VC is scheduled in RR. Fairness across multiple GPU traffic streams can be achieved. However, naive scheduling will lead to contention on the PCIe interconnect, resulting in long GPU stalls.

3.2 Priority-Based Scheduling for Throughput

To mitigate bandwidth contention, we propose a dynamic priority-based PCIe scheduling policy. The key idea is to hide the memory transfer latency by overlapping it with kernel execution of different GPUs. To enable this capability, we increase the priority of some memory transfers. These memory transfers can be completed without interruption, allowing the associated GPU kernel can start executing as soon as the transfer is complete. Low-priority data transfers can be performed while higher-priority tasks are already in execution. As data-center workloads often exhibit repeated *"copy-then-execute"* patterns, our policy can effectively reduce data transfer interference among different tenants.

We develop a Throughput-Oriented Scheduler (TOS) in the driver, as shown on the left side of Fig. 4. TOS uses *Strict Priorities* for VC arbitration, where *VC7* has the highest priority and *VC0* has the lowest priority. The arbiter always handles requests from highest priority VCs first.

We classify packets into *two* types ($N_T = 2$): 1) request packets ($ID_T = 0$, read packets and write packets) and 2) response packets ($ID_T = 1$, read response packets). TOS always gives response packets higher priority. By granting responses higher priority, data transfer transactions can be completed sooner, enabling programs to start running on the GPU sooner.

Priorities are assigned to the packets of different tasks by using TC/VC remapping. Priorities depend on the size of the remaining data transfer associated with each memory transfer request issued by different tasks. Inspired by the idea of "small flow first" in

TABLE 1
Benchmarks

APP ID	Abbr.	Workload	Size Per Iteration (KB)
1	SC	Simple Convolution [10]	328.4
2	MM	Matrix Multiplication [10]	128
3	MT	Matrix Transpose [10]	1024
4	AES	AES-256 Encryption [11]	65.2
5	FIR	FIR Filter [11]	256.1
6	KS	KMeans Clustering [11]	131.1
7	MP	Maxpooling [12]	64
8	RL	Relu [13]	32

network, tasks with fewer data to be transferred are granted higher priority, which should reduce GPU stall time and achieve higher execution throughput. A remaining data-transfer size list is managed in the TOS to keep track of the remaining data for each task. When a new memory transfer command is received, the TOS ranks the priorities by sorting the remaining data sizes for each task and updating the TC/VC remapping. For tasks with lower priorities, it is possible that starvation may happen in some extreme cases. Thus, we set a threshold x empirically for each VC. If the head request in the VC is blocked for x cycles, it will be handled immediately to avoid starvation.

Compared with RR, TOS reduces the bandwidth contention for all users under multi-tenancy. However, if a task with a strict QoS requirement is given low priority by the PCIe scheduler, the QoS goal will be violated due to excessive wait time. In Section 3.3, we propose a priority switching policy to achieve the QoS goal for those tasks with lower priority.

3.3 Priority Switching for Semi-QoS Management

Priority-based scheduling will increase the PCIe and GPU utilization, and hence improve the throughput of the entire multi-tenant GPU system. However, in many domains, some applications may have QoS requirements (e.g., the application should complete within a deadline). When such a workload is hosted in a multi-GPU architecture, priority-based PCIe scheduling alone is insufficient since the QoS task may need more resources than other tasks.

For QoS tasks, the goal of scheduling is to achieve the QoS target. Once the QoS target can be met, the scheduler will then attempt to maximize the global throughput. In this work, we define the latency of each workload as the QoS target. The true QoS requirement will account for end-to-end latency, which includes the GPU execution time. We assume the OS will be able to define a set of partial QoS goals for various resources. Our scheduling policy will provide semi-QoS management as part of the QoS management deployed in the OS. In this work, our partial QoS goal is thus the memory transfer time that is spent on the PCIe bus.

We introduce a QoS-Oriented Scheduler (QoOS), as shown on the right side of Fig. 4 to implement semi-QoS management. To predict whether the memory transfer can complete in time, QoOS keeps track of the transfer time of each command for the QoS tasks associated with each GPU on the PCIe interconnect. As the remaining data size can be collected by the TOS, and the PCIe bandwidth is known, we can predict the time to send the remaining data for this command. We compute the highest priority using Equation (3)

$$T_{trans} = B_R / BW, \tag{3}$$

where T_{trans} is the time left to transfer the data, B_R is the remaining number of byte to transfer, and BW is the PCIe bandwidth.

Initially, priorities are ranked by data sizes of different tasks, in order to achieve high throughput for the entire system. For QoS tasks, we set a deadline for each iteration of the task (recall that we run multiple iterations of the same workload for testing purposes) to approximate the QoS requirements in a real scenario.

Once QoOS predicts the T_{trans} is too long and will miss the deadline, the priority of the current task is raised to the highest level to try to meet the QoS goal and updating the TC/VC remapping.

4 Preliminary Evaluation

We extend MGPUSim [13] to evaluate our priority-based scheduling Scheme. The modeled multi-GPU system consists of a CPU with four AMD R9Nano [14] GPUs. In this work, we run two kinds of tasks, launched by two users to model multi-tenant sharing. For experimental purposes, a task with a larger data size is assumed to be the QoS task, while the task with smaller data size is assumed as a non-QoS task. The deadline of the memory transfer time is set as *3 times* the memory transfer time without bandwidth contention. The 8 workloads we used to evaluate the proposed solution are from the *AMDAPPSDK* [10], *Hetero-Mark suite* [11] and *clCaffe* [12] suites, as shown in Table 1. We select the problem size to find a good trade-off between the simulation time and common use-cases of applications.

To explore the concurrent execution of tasks, we generate 28 (8*7/2) combinations in total. Each task is repeatedly launched on a GPU until the total measured time exceeds 0.3 seconds. To evaluate system throughput, we use the total number of iterations executed during 0.3 seconds, normalized to the total number of iterations running on a single GPU system, as our figure of merit. The number of total iterations for all workloads is 81.75 on average, thus it is large enough to avoid the effect of the job arrival skew.

Four PCIe scheduling policies are implemented, including (*RR*), Priority scheduling with large data size first (*Priority_L*), Priority scheduling with small data size first (*Priority_S*) and QoS support of Priority scheduling with small data size first (*QoS*).

Fig. 5 shows the total multi-GPU throughput for four PCIe scheduling policies, normalized to RR. We make three observations. First, *Priority_S* achieves the highest performance and outperforms *RR* by 7.6 percent on average. Second, giving high priority to memory commands that possess a smaller data size is better than giving high priority to memory commands with larger data sizes, as the idle time of the GPUs is reduced. Third, although *QoS* cannot perform as well as *Priority_S* due to QoS requirements, it still outperforms *RR* by 5.3 percent on average.

Fig. 6 shows the performance of two concurrent tasks. Task 1 has a smaller data size, while task 2 has a larger data size. Both tasks in

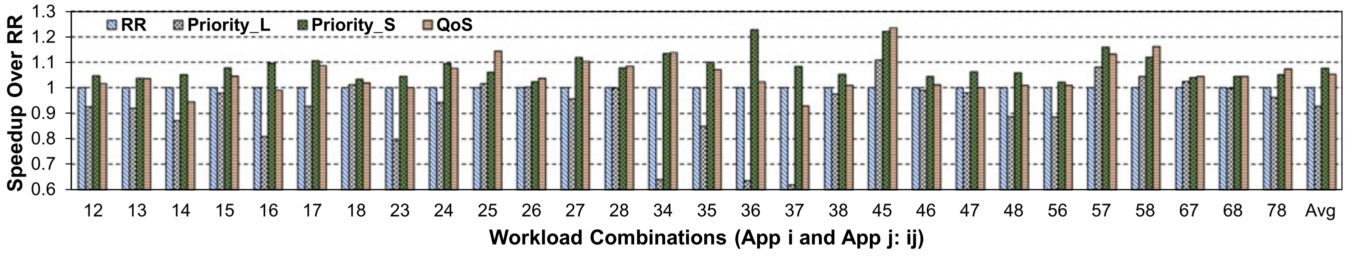

Fig. 5. Total throughput of the tested multi-GPU system.

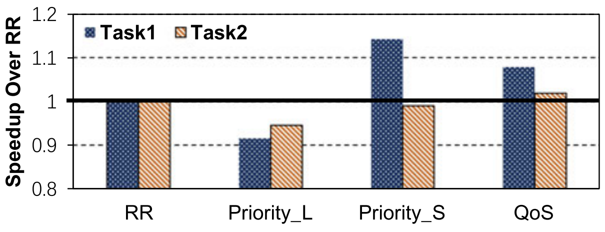

Fig. 6. Throughput of two concurrent tasks running on the Multi-GPU system.

Priority_L perform worse than in *RR* due to long stall times, waiting for large datasets to complete the transfer. In *Priority_S*, although task 1 can outperform *RR* by 14.2 percent, the performance of task 2 is decreased by 1 percent due to the low priority of this task. In priority scheduling policy With QoS support, though there is no complete fairness, both tasks perform better than *RR*. Moreover, the QoS achievement rate can be higher.

5 Conclusion

In this paper, we propose a runtime-defined priority-based PCIe scheduling policy to improve system throughput. Experimental results show that system throughput can be improved by 7.6 percent on average with the priority-based PCIe scheduling as compared with the Round-Robin-based PCIe scheduling. In future work, we will focus on QoS management in multiple nodes with NIC and NFS/disk access.

Acknowledgments

This work was supported in part by Grants from NSF CNS-1525412; NSF of China (61672526); Research Project of NUDT (ZK17-03-06) and Science and Technology Innovation Project of Hunan Province (2018RS3083).

References

[1] Amazon, “Amazon EC2 P3 instances: Accelerate machine learning and high performance computing applications with powerful GPUs,” 2019. [Online]. Available: https://aws.amazon.com/ec2/instance-types/p3/
[2] Y. Sun *et al.*, “Managing access to a resource pool of graphics processing units under fine grain control,” U.S. Patent App. 16/287,719, Jun. 27, 2019.
[3] NVIDIA, “Nvidia DGX-2 system,” 2018. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-2/
[4] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, “Analysis of large-scale multi-tenant GPU clusters for DNN training workloads,” in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2019, pp. 947–960.
[5] D. Sengupta, R. Belapure, and K. Schwan, “Multi-tenancy on GPGPU-based servers,” in *Proc. 7th Int. Workshop Virtualization Technol. Distrib. Comput.*, 2013, pp. 3–10.
[6] W. Wei *et al.*, “AI matrix-synthetic benchmarks for DNN,” 2018, *arXiv preprint arXiv:1812.00886*. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1812/1812.00886.pdf
[7] A. Arunkumar *et al.*, “MCM-GPU: Multi-chip-module GPUs for continued performance scalability,” *ACM SIGARCH Comput. Architecture News*, vol. 45, pp. 320–332, 2017.
[8] U. Milic *et al.*, “Beyond the socket: NUMA-aware GPUs,” in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2017, pp. 123–135.
[9] V. Young, A. Jaleel, E. Bolotin, E. Ebrahimi, D. Nellans, and O. Villa, “Combining HW/SW mechanisms to improve NUMA performance of multi-GPU systems,” in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2018, 339–351.
[10] AMD, “AMD APP SDK 3.0 getting started,” 2017. [Online]. Available: http://developer.amd.com/wordpress/media/2013/12/AMD_APP_SDK_GettingStartedGuide.pdf
[11] Y. Sun *et al.*, “Hetero-mark, a benchmark suite for CPU-GPU collaborative computing,” in *Proc. IEEE Int. Symp. Workload Characterization*, 2016, pp. 1–10.
[12] J. Bottleson, S. Kim, J. Andrews, P. Bindu, D. N. Murthy, and J. Jin, “clCaffe: OpenCL accelerated caffe for convolutional neural networks,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2016, pp. 50–57.
[13] Y. Sun *et al.*, “MGPUSim: Enabling multi-GPU performance modeling and optimization,” in *Proc. 46th Int. Symp. Comput. Architecture*, 2019, pp. 197–209.
[14] AMD, “AMD radeon R9 nano, world’s smallest and most power-efficient enthusiast graphics card, brings 4k gaming to the living room,” 2015. [Online]. Available: https://www.amd.com/en/press-releases/amd-radeon-r9-nano-2015aug27