PCIe Bandwidth-Aware Scheduling for Multi-Instance GPUs

Yan-Mei Tang
National Tsing Hua University
Hsinchu, Taiwan
ymtang@lsalab.cs.nthu.edu.tw

Wei-Fang Sun
NVIDIA AI Technology Center
Santa Clara, CA, USA
johnsons@nvidia.com

Hsu-Tzu Ting
National Tsing Hua University
Hsinchu, Taiwan
hsutzu.ting@lsalab.cs.nthu.edu.tw

Ming-Hung Chen
IBM Research
New York, USA
minghungchen@ibm.com

I-Hsin Chung
IBM Research
New York, USA
ihchung@us.ibm.com

Jerry Chou
National Tsing Hua University
Hsinchu, Taiwan
jchou@lsalab.cs.nthu.edu.tw

ABSTRACT

The increasing computational power of GPUs has driven advancements across various domains, especially in scientific computing and machine learning. However, lighter workloads often do not fully utilize a GPU's capacity, leading to inefficiencies. The Multi-Instance GPU (MIG) feature in NVIDIA A100 GPUs addresses this issue by allowing a single GPU to be divided into multiple, smaller, isolated instances, thus improving resource allocation for multi-tenant environments. While MIG provides enhanced isolation and predictable performance, we observed that PCIe bandwidth remains a shared resource, which can lead to contention when multiple instances require high bandwidth, such as running concurrent machine learning inference tasks. In this paper, we identify and address this issue, being among the first to demonstrate PCIe bandwidth contention across MIG instances in tasks with high bandwidth demands. We propose a PCIe bandwidth-aware MIG scheduler that predicts and mitigates contention by preventing simultaneous scheduling of bandwidth-intensive jobs on the same GPU. Our scheduler leverages a performance model to quantify PCIe contention severity, enabling more efficient scheduling decisions. Experimental results show that the proposed scheduler reduces job completion times by approximately 18%, improving GPU resource utilization in both real-world and larger-scale simulated environments.

CCS CONCEPTS

• **Hardware** → **Emerging technologies**; • **Computer systems organization** → *Multicore architectures*; • **Networks** → *Network performance evaluation.*

ACM Reference Format:
Yan-Mei Tang, Wei-Fang Sun, Hsu-Tzu Ting, Ming-Hung Chen, I-Hsin Chung, and Jerry Chou. 2024. PCIe Bandwidth-Aware Scheduling for Multi-Instance GPUs. In *Proceedings of Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia '25).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

HPCAsia '25, February 19–21, 2025, Hsinchu, Taiwan

ACM ISBN 979-8-4007-1335-4…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

GPUs provide powerful computing capability, driving significant advancements in scientific computing and machine learning. They have also facilitated the development of numerous AI services, enhancing convenience in everyday life. As GPU vendors continue to roll out new products, the computational capabilities of GPUs become increasingly powerful. However, users employing lighter models may be unable to fully leverage the computational capacity of a GPU, leading to resource wastage [13].

To address this issue, NVIDIA has proposed Multi-Instance GPU (MIG) [8], which divides a single GPU into multiple smaller instances (or MIG instances), through hardware support. This enables finer-grained resource allocation and utilization of the GPU, providing better hardware isolation and potentially improving overall performance [7]. MIG has several use cases, such as providing hardware isolation for cloud computing services. It allows users to securely rent a portion of GPU resources while ensuring critical features such as fault isolation and distinct address spaces, both of which are essential in multi-tenant environments. In addition, using MIG for services with Service Level Objectives (SLOs) improves predictability of execution times and reduces tail latency compared to software-based isolation. This improvement arises because software-based isolation can cause interference between concurrently running jobs, making execution times difficult to predict [15]. MIG also offers a reconfiguration feature, allowing users to dynamically reallocate the size of MIG instances based on varying task demands in cloud environments. This capability sets MIG apart from traditional heterogeneous multi-GPU environments, where resource allocation is restricted to scheduling tasks on GPUs of different sizes. In contrast, MIG offers a more fine-grained approach, achieving better resource utilization.

Although MIG effectively achieves isolation across various hardware resources, some resources, like PCIe bandwidth, remain shared. Shared bandwidth can become a bottleneck when multiple MIG instances require significant PCIe bandwidth, resulting in increase in data transfer wait times and decrease in performance. For example, users with limited budgets who want to run multiple ML inference jobs concurrently can partition a GPU into several MIG instances. However, if these jobs consume significant PCIe bandwidth, contention issues may arise, leading to longer inference runtime and reduced throughput. Moreover, MIG reconfiguration is only possible when sufficient contiguous space is available [8]. To fully leverage the benefits of reconfiguration, the scheduling

design must account for both the constraints of reconfiguration and the desired sizes of the MIG instances being created.

Existing works on GPU scheduling often focus on heterogeneous GPU scheduling, which takes account the varying computational capability of different GPUs. However, these works do not utilize the reconfiguration flexibility provided by MIG. Other MIG schedulers allocate MIG resources based on computational workload and memory usage but overlook the potential impact of PCIe bandwidth contention on performance.

In this paper, we propose a PCIe bandwidth-aware MIG scheduler, an online MIG scheduler designed to prevent PCIe bandwidth contention. We design a performance model to quantify the severity of PCIe contention, enabling our scheduler to predict potential PCIe contention. The scheduler then avoids scheduling jobs with high PCIe bandwidth demands (referred to as '*PCIe-bound jobs*' hereafter) on the same GPU, mitigating PCIe contention issues. Our results show an approximately 18% reduction in aggregated job completion time in both real-world experiments and larger-scale simulations.

The remainder of this paper is organized as follows: Section 2 reviews related work on GPU scheduling, with a focus on the limitations of existing methods in addressing PCIe bandwidth contention and Multi-Instance GPU (MIG) environments. Section 3 describes preliminary experiments that illustrate the sharing behavior of PCIe bandwidth contention among MIG instances, providing motivation for our proposed approach. In Section 4, we formalize the PCIe bandwidth-aware scheduling problem, introduce a performance model, and present our proposed scheduler designed for MIG GPU environments. Section 5 covers the evaluation setup and results, comparing the performance of our scheduler with a baseline that is not PCIe bandwidth-aware, demonstrating reductions in total job completion time and improved efficiency. Finally, Section 6 concludes with a summary of the main results.

The contributions of our work are summarized as follows:

- To the best of our knowledge, this is the first work to investigate and mitigate PCIe bandwidth contention between MIG instances in real GPU products.
- We demonstrate the occurrence of PCIe bandwidth contention through real-world tasks and observe that it results in increased job execution times, leading to overall performance degradation.
- We introduce an online PCIe bandwidth-aware scheduler to mitigate the PCIe bandwidth contention issue while considering MIG instance size and reconfiguration constraints.
- We demonstrate that our proposed scheduler outperforms a baseline scheduler, which does not account for PCIe bandwidth contention, by approximately 18% in both simulations and real GPU hardware experiments.

2 RELATED WORK

2.1 Multi-Instance GPU (MIG) Scheduler

Multi-instance GPU (MIG) allows multiple jobs to run concurrently on a single GPU with a predefined configuration specifying how to partition MIG instances. Both the configuration and the MIG allocation to jobs can significantly impact efficiency, emphasizing the importance of scheduling. MISO [7] aims to determine the optimal MIG partition allocation for a given set of jobs. However, exhaustively evaluating all possible MIG partition allocations results in prohibitive overhead due to the large number of GPU reconfigurations and resets required. To mitigate this, MISO estimates the performance of various MIG configurations by running jobs in the more flexible MPS mode, adjusting the GPU computing resource percentage allocated to each job, and then selecting the partition allocation with the best predicted performance. CASE [10] shows that relaxing the isolation property of jobs which do not have strict isolation requirements can lead to higher system throughput. CASE achieves this by utilizing a MIG instance, which can be dedicated to serve a single isolated job, or shared among multiple non-isolated jobs. However, both MISO and CASE solely focus on compute-bound tasks, overlooking the potential performance impact of PCIe bandwidth contention.

2.2 PCIe Bandwidth-Aware Schedulers

PCIe bandwidth is a limited resource that schedulers must take into account. CuMAS[2] treats the GPU, PCIe uplink, and PCIe downlink as three separate resources that are independently allocated to CUDA requests from applications. By reordering these CUDA requests using `LD_PRELOAD`, CuMAS effectively overlaps data transmission with kernel execution, reducing wait times and enhancing system throughput. Baymax[4] reorders task and kernel execution sequences and limits the number of tasks utilizing PCIe bandwidth simultaneously, thereby eliminating performance degradation caused by PCIe bandwidth contention and ensuring compliance with QoS regulations. However, these two studies overlook the strict hardware isolation requirements in cloud computing services. In contrast, our approach utilizes task-level scheduling rather than scheduling lower-level primitives such as CUDA kernels. This method is less dependent on low-level control assumptions and offers a more generalized solution, which is essential for maintaining proper hardware isolation.

2.3 Heterogeneous GPU Scheduler

A heterogeneous GPU scheduler is essential for a cluster with multiple types of GPU. Hydra[14] is an online scheduler for heterogeneous GPUs, designed to enhance system efficiency while maintaining deadline awareness. It predicts the runtime of each job across different GPU types and selects the optimal job placement by using pruning techniques to explore all possible scheduling combinations efficiently. $Gandiva_{fair}$ [3] ensures that each user receives the expected amount of computing power in a heterogeneous GPU cluster while improving the overall efficiency of the system. It utilizes migration and resource trading between users to maintain the expected performance of deep learning training jobs and to improve overall system utilization. However, these heterogeneous GPU schedulers do not consider the reconfiguration capability offered by MIG, which could provide greater flexibility in GPU sharing.

2.4 MIG Applications

Previous works have demonstrated the advantages of GPU sharing. Nexus [11] highlights that many applications involve running multiple Deep Neural Network (DNN) models simultaneously to deliver

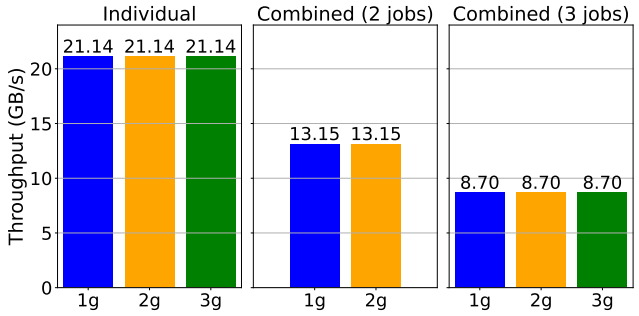


Figure 1: CUDA memory copy throughput on different size of MIG instance. 1g, 2g, 3g represent 1g.5gb, 2g.10gb, 3g.20gb MIG instances respectively. It is observed that when PCIe bandwidth contention occurs, the bandwidth is distributed equally among the MIG instances.

a complete services. For example, ensemble models [12] aggregate several sub-models to enhance overall accuracy. Since the training workload for each model is relatively low, MIG is ideal for concurrently training multiple sub-models on a single GPU. Furthermore, MIG has the potential to deliver performance improvements over workloads running without MIG [8]. Specifically, the combined performance of multiple small MIG instances on a single GPU exceeds that of using the entire GPU directly [6].

3 PRELIMINARY EXPERIMENTS

In this section, we investigate the PCIe bandwidth contention issue on NVIDIA A100 GPUs through two preliminary experiments. These experiments are conducted on a single node equipped with an NVIDIA A100-PCIe-40GB GPU installed in a PCIe 4.0 slot.

3.1 PCIe Bandwidth Sharing Behavior

In this experiment, we aim to demonstrate the existence of PCIe bandwidth contention between MIG instances and confirm that PCIe bandwidth is equally distributed among multiple MIG instances when contention occurs. These two findings are crucial for the scheduler design discussed in the subsequent sections.

To verify this, we developed a simple copy program that repeatedly transfers 5GB of floating-point data from host to device. We executed this program simultaneously on `1g.5gb`, `2g.10gb`, and `3g.20gb` MIG instances on the same GPU and profiled the process using NVIDIA Nsight System [9].

As shown in Fig. 1, both jobs experienced performance degradation due to PCIe bandwidth contention, with each job showing identical memory copy throughput. This confirms that PCIe bandwidth contention exists between MIG instances on the same GPU, leading to reduced performance and longer data transfer times. In addition, we observed that PCIe bandwidth is equally shared between MIG instances regardless of their sizes. This suggests that the allocation of PCIe bandwidth depends on the number of competing MIG instances rather than the size of the individual instances.

3.2 PCIe Bandwidth Contention in Inference Jobs

To demonstrate that PCIe bandwidth contention can occur in real-world scenarios, we conducted experiments with inference jobs that are heavily dependent on PCIe bandwidth.

In this experiment, we utilized the Bloom560m model [5] with ZeRO-Inference [1], a technique designed to run large models on limited GPU resources. ZeRO-Inference operates by storing model weights in CPU memory and transferring them to the GPU only when needed for computation. This approach results in substantial data transfers between the CPU and GPU during inference, as model weights are loaded layer by layer. As a result, the execution time of this job is heavily dependent on available PCIe bandwidth, making it a PCIe-bound job. We ran a Bloom560m inference job on each 1g MIG instance, varying the number of active 1g MIG instances on the same GPU, and measured the resulting inference throughput.

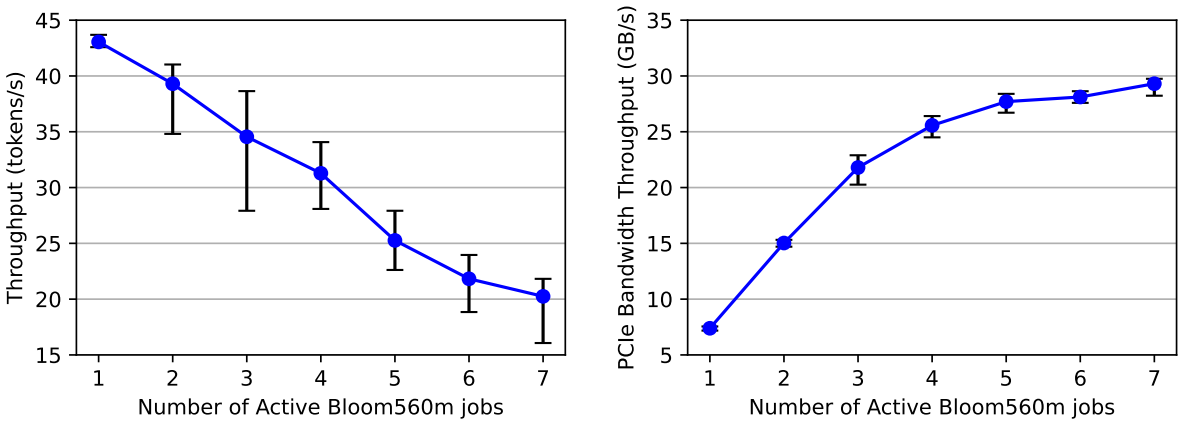
When running a single Bloom560m with ZeRO-Inference on a 1g MIG instance, the job consumes approximately 30% of the total PCIe bandwidth available to the GPU. As shown in Fig. 2a, we observed a noticeable decline in throughput when the number of active MIG instances exceeded four, indicating the threshold at which PCIe bandwidth becomes fully saturated. Furthermore, Fig. 2b illustrates that the GPU's PCIe bandwidth usage gradually approaches its maximum capacity as more jobs run concurrently. Once the number of active jobs surpasses four, the PCIe bandwidth becomes completely saturated.

This experiment highlights the presence of PCIe bandwidth contention in practical scenarios. For instance, users constrained by the size of their allocated MIG instance may rely on offloading techniques to execute models, leading to significant PCIe bandwidth consumption. Running multiple PCIe-bound jobs simultaneously exacerbates contention, resulting in degraded performance.

4 METHODOLOGY

As discussed in the previous section, the job execution time on MIG GPU needs to consider two key factors. One is the size of the MIG slice, because a job might have a shorter execution time when running on a larger MIG slice. The other one is the PCI contention of the GPU, because a job may suffer performance degradation when the PCI bandwidth is over-subscribed by the co-located jobs on the same GPU. As a result, in order to minimize the average job completion time (JCT) of a MIG GPU cluster, it is crucial to decide the proper MIG slice size and the GPU location of each job by the scheduler. However, most of the scheduling works on MIG GPU cluster only focus on addressing the first problem by finding the proper MIG configuration according to the computing demand, little attention has been paid to the second problem for addressing the bandwidth contention problem of MIG GPUs.

To address the bandwidth contention issue of MIG GPUs, we first construct a performance model to estimate the job slowdown based on the performance profiling results of jobs, then we propose an online scheduling algorithm to place the jobs according to the predicted job execution time. Finally, we address the MIG GPU configuration problem by re-partitioning a MIG GPU when it is idle for minimizing the job wait time. Our PCIe Bandwidth



(a) Throughput versus number of Bloom560m jobs.

(b) Total PCIe bandwidth usage versus number of jobs.

Figure 2: PCIe bandwidth contention behavior when running multiple Bloom560m jobs. The error bars represent the 0.975 quantile and 0.025 quantile of the respective values.

contention aware performance model and scheduler are detailed in the following subsections.

4.1 Performance Model

We first develop a performance model based on profiling data jobs to quantify the slowdowns caused by PCIe contention. Noted, in practice, we don't need to do the profiling and modeling for every single job individually, because jobs may shared the similar performance characteristics among each other, especially for the jobs running the same application with different running parameters as shown in our experiments. This performance model then enables us to estimate the potential slowdowns that may occur when multiple PCIe-bound jobs are executed on the same GPU, and guide the online scheduler to reduce job completion as described in the next subsection. The symbols used in this paper are listed in Table 1.

Job Profiling. Prior to scheduling, the maximum PCIe bandwidth available for each GPU (denoted as $PCIe\_max$), is measured. For the j-th job, its PCIe bandwidth demand is represented as $PCIe\_demand(j)$ (with non-PCIe-bound jobs having a PCIe demand of 0), its base processing throughput when running without PCIe bandwidth contention (denoted as $base\_throughput(j)$), and its performance sensitivity to allocated PCIe bandwidth α_j are all derived from the job profiling data. Profiling is done by running several instances of the same job type on the same GPU, each within its own MIG instance. We calculate α_j using linear regression on the observed runtimes and allocated PCIe bandwidth. Specifically, when we run n instances simultaneously on the same GPU, we aim to learn the following relationship:

$$\frac{runtime(j,n)}{base\_runtime(j)} = \alpha_j \times \frac{PCIe\_demand(j)}{PCIe\_max/n}, \tag{1}$$

where $runtime(j,n)$ and $base\_runtime(j)$ represent the profiled runtimes for n instances and a single instance, respectively, on the same GPU.

Eq. (1) is derived from two key observations. First, as outlined in Section 3.1, PCIe bandwidth is equally divided among MIG instances (yielding $PCIe\_max/n$ per instance). This equal distribution may cause slowdowns when the PCIe demand ($PCIe\_demand(j)$) surpasses the allocated bandwidth per instance. Second, our profiling results, discussed in Section 5.2, reveal a linear relationship between the slowdown (left-hand side of Eq. 1) and the number of jobs (n). Hence, we introduce a single performance sensitivity value α_j to model the slowdown in our current proposed model. A more complex and accurate model could certainly be considered for applications with more diverse runtime behavior, but we will leave it as our future work.

Slowdown Prediction. For the j-th job, scheduled to run on the g-th GPU, we estimate the slowdown based on the number of PCIe-bound jobs running on the GPU (denoted as $PCIe\_jobs(g)$) at current time, given by:

$$slowdown(j,g) = \max(1, \alpha_j \times \frac{PCIe\_demand(j)}{PCIe\_max/PCIe\_jobs(g)}) \tag{2}$$

This equation ensures that the slowdown is at least 1 and scales proportionally with the number of PCIe-bound jobs that compete for PCIe bandwidth with the j-th job. Based on estimated slowdown and job information, we can predict the remaining execution time of each job as follows:

$$exec\_time(j,g) = \frac{remain\_work(j)}{base\_throughput(j)} \times slowdown(j,g), \tag{3}$$

where the $remain\_work(j)$ denotes the remaining work for the j-th job at current time, which is estimated in terms of tokens and kept track by the scheduler, and the $exec\_time(j,g)$ is the remaining execution time predicted by the scheduler.

With this performance model, we can estimate the execution times for both PCIe-bound and non-PCIe-bound jobs, allowing the prediction of their expected completion times when assigned to a specific GPU. These predictions are then utilized to guide scheduling decisions, as detailed in the following section.

Table 1: The list of symbols used in this paper.

Symbol	Description	Unit	Context
$PCIe\_max$	Maximum PCIe bandwidth per GPU	GB/s	Job Profiling
$PCIe\_demand(j)$	PCIe bandwidth demand of the j-th job when running exclusively on the GPU	GB/s	
$base\_throughput(j)$	Base processing throughput of the j-th job when running exclusively on the GPU	tokens/s	
α_j	Performance sensitivity of the j-th job	-	
$base\_runtime(j)$	Runtime of the j-th job when running exclusively on the GPU	second	
n	Number of job instances running on the GPU during profiling	-	
$runtime(j, n)$	Runtime of the j-th job when running n instances on the GPU	second	
$remain\_work(j)$	Estimated remaining work for the j-th job at current time	tokens	Scheduling
$slowdown(j, g)$	Estimated slowdown for the j-th job if scheduled on the g-th GPU at current time	-	
$exec\_time(j, g)$	Estimated remaining execution time for the j-th job if scheduled on the g-th GPU at current time	second	
$PCIe\_jobs(g)$	Number of PCIe-bound jobs on the g-th GPU at current time, including the job currently being scheduled	-	
$delay\_threshold$	Maximum allowable predicted slowdown for a job to be scheduled on a GPU	-	User-defined
$wait\_time\_threshold$	Maximum duration a job can wait before the scheduler bypasses $delay\_threshold$ constraint.	second	

4.2 PCIe Bandwidth-Aware MIG Scheduler

Given a MIG GPU cluster with arriving jobs, the PCIe bandwidth-aware scheduler is activated whenever a new job arrives or a job completes. It repeatedly iterates through each job in the waiting queue until no more jobs can be scheduled in the current round.

As we focus on the scheduling problem caused by the bandwidth contention, here we consider the requested MIG slice size of a job are fixed and given by the users and jobs are prefer to be scheduled to the MIG slice with the requested size. However, the scheduler still have to solve two problems caused by the requested MIG size. First, if there are multiple GPUs that have available MIG slice for the job, which GPU should be chosen? Second, if there are no GPUs that can provide the requested MIG slice size for the job, when and how to reconfigure MIG GPUs, so the job wait time can be minimized?

With our predicted job time information, we could easily address the first question by selecting the GPU that causes the minimum job slowdown time. But for the second question regarding reconfiguration, we have to applied the following scheduling principals: (1) Reconfiguration occurs only on the idle GPUs without any running jobs. This is because the reconfiguration operations may reset the whole GPU and disrupt the running jobs based on the existing MIG reconfiguration mechanism. (2) To gain more opportunities for GPU reconfigurations, we will avoid extending the running end time of a GPU which is determined by the latest job completion time on a GPU. (3) We will only reconfigure a GPU for jobs that cannot find the matching MIG slices from the existing configurations. (4) When a job waits too long without finding its requested MIG slice, we will relax our scheduling constraint on the slice size from exact-match to best-fit.

In sum, our scheduling algorithm is consisted of the following four main stages:

(1) Update Internal Records. Upon activation, the scheduler updates its internal records, tracking which jobs are running on each GPUs, which jobs are skipped in the current round due to unavailability of required MIG instance size or anticipated PCIe bandwidth contention, and adjusting each job's $remain\_work(j)$ based on elapsed time since the last activation. This information is essential for the subsequent scheduling stages.

(2) Reconfigure GPUs. The scheduler then verifies if any GPU has been marked for reconfiguration in previous iterations. Reconfiguration occurs only when all jobs on the GPU has completed (i.e., GPU is idle). We do not consider reconfigurations on the GPUs with running jobs because it may cause jobs interruptions based on the existing MIG reconfiguration mechanism. Hence, our strategy is to only reconfigure the empty GPU for the jobs in the waiting queue that couldn't find available slices previously. In order to minimize wait time and maximize GPU utilization. We will reconfigure the empty GPU, so that the most number of waiting jobs can be scheduled on the reconfigured GPU.

(3) Select the Most Suitable GPU. A multi-stage strategy is employed to select the most suitable GPU for the current job being scheduled. First, GPUs without MIG instances with the required size are filtered out, as jobs are only scheduled to MIG instances with the requested size. If all GPUs are filtered out, the scheduler marks the GPU with the earliest estimated end time, avoiding assigning new jobs to it so it can be freed for reconfiguration as quickly as possible. However, if a job's end time does not exceed the marked GPU's estimated end time, the scheduler may assign it to the marked GPU to maximize resource utilization. Second, the scheduler sorts the GPUs according to the slowdown calculated by Eq.(2), prioritizing the GPUs with lower slowdown values. If multiple GPUs have the same slowdown, the scheduler selects the best-fit to consolidate jobs on fewer GPUs. This approach reduces fragmentation and maintains availability for future job reconfiguration.

(4) Run or Skip the Job. The final stage decides whether to run the job on the GPU selected in the previous stage based on $delay\_threshold$ and $wait\_time\_threshold$. These two parameters are predefined by the users. $delay\_threshold$ prevents scheduling on the GPU with a significant slowdown due to PCIe bandwidth contention. The scheduler skips the job when the selected GPU has a slowdown exceeding $delay\_threshold$. The only exception is that the

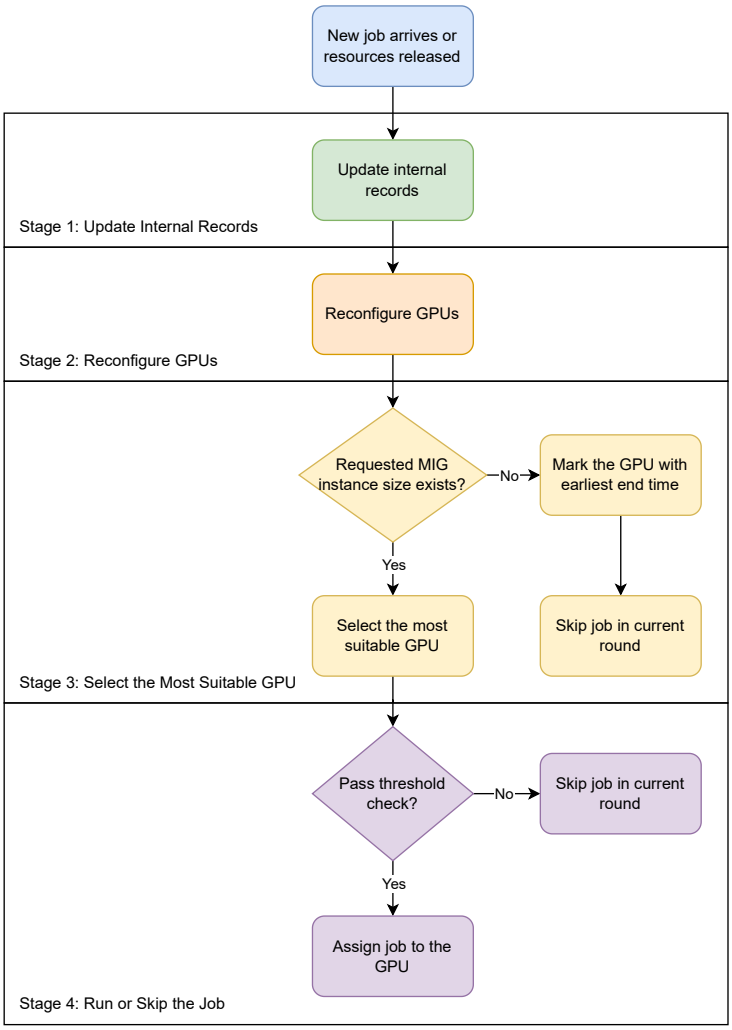


Figure 3: Execution flow for each iteration of the PCIe bandwidth-aware scheduler.

job is starving. *wait_time_threshold* prevents the job from waiting indefinitely. If the job's wait time exceeds *wait_time_threshold* , the GPU is allowed to bypass the *delay_threshold* constraint.

5 EVALUATION

5.1 Setup

Testbed. We conduct experiments on a single node equipped with 256 AMD EPYC 7742 CPUs and 4 NVIDIA A100-PCIe-40GB GPUs. The system supports PCIe 4.0, whose achievable maximum PCIe bandwidth capacity is around 30.08 GB/s. All the GPUs are directly connected to the root complex, so there would be no PCIe bandwidth contention between them. We initially partition A100 GPUs into `1g.5gb` MIG instances, and the partitioning can be reconfigured according to the incoming job requests based on our scheduling algorithm.

Simulator. To measure the improvement in job completion time in an environment beyond our testbed, we design the simulator to model the execution of jobs. It can simulate the cluster with any number of GPUs and GPUs with different maximum numbers of MIG instances. The job execution time on GPU is based on the performance model.

Workload. Our workload includes PCIe-bound jobs and non-PCIe-bound jobs as shown in Table 2. Bloom560m and Bloom7b1 are PCIe-bound jobs, while Resnet50 is a non-PCIe-bound job. Their PCIe demand and α_j are obtained in Section 5.2.

Baseline Scheduler. The baseline scheduler is unaware of PCIe bandwidth contention and assigns jobs to a GPU as long as sufficient MIG resources exist. If multiple GPUs have enough resources, the scheduler selects the first one according to the order of GPU IDs.

Table 2: Workloads evaluated in the experiments. The PCIe demand for ResNet50 is marked as 0, as it is not a PCIe-bound workload.

Job Type	Resnet50	Bloom560m	Bloom7b1
Model Size	25.6M	560M	7.07B
PCIe Demand (GB/s)	0	5.7	17.65
α	-	1.25	1.07
PCIe-bound Job	No	Yes	Yes

5.2 Profiling

Profiling is done by running several instances of the same job type on the same GPU, aiming to measure $PCIe\_demand(j)$ and $runtime(j, n)$, and to further derive the value of α_j. Fig. 4 illustrates the profiling results for Bloom7b1 and Bloom560m. The x-axis indicates the severity of PCIe contention, while the y-axis shows the job's slowdown. The figure validates Eq. (1), showing that the slowdown is linearly correlated with PCIe bandwidth contention, with the slope given by α_j. We derived the α_j of Bloom7b1 and Bloom560m to be 1.07 and 1.25, respectively, and these values are used in the subsequent experiments.

5.3 Testbed Evaluation

We measured the job execution time and job completion time (JCT) on our testbed to demonstrate the effectiveness of our PCIe bandwidth-aware scheduler. Fig. 6 shows the JCT of our PCIe bandwidth-aware scheduler and the baseline scheduler. JCT is the sum of job wait time and job execution time. Our scheduler consistently outperforms the baseline scheduler across all different PCIe-bound job ratio settings. Moreover, the benefits of our scheduler become even more significant as the proportion of PCIe-bound jobs increases. When the PCIe-bound jobs account for 20%, the PCIe bandwidth-aware scheduler is slightly better than the baseline scheduler. While the PCIe-bound jobs are as much as 60%, the aggregated JCT is reduced by up to 18%. Next, we plot the Kernel Density Estimate (KDE) to analyze the jobs execution time when PCIe-bound jobs account 60% of the workload in Fig 5. There are 50% more jobs with low execution time (50 seconds) scheduled by the PCIe bandwidth-aware scheduler than the baseline scheduler. That is because our scheduler effectively reduces PCIe bandwidth contention compared to the baseline scheduler, resulting in shorter execution times for PCIe-bound jobs.

5.4 Simulation Results

To assess the scalability of our scheduler, we developed a simulator designed to closely replicate real-world evaluation results. This simulator allows us to evaluate the scheduler in a larger-scale system, and incorporating hypothetical GPUs with varying maximum numbers of MIG instances per GPU.

Simulator Validation. We begin by validating our simulator against our testbed and subsequently present the evaluation results. The simulator emulates a node equipped with four A100 GPUs, which is identical to our testbed, and use the same workload as in the testbed evaluations for the validation experiment. Fig. 7 shows the normalized JCT reported by our simulator, which closely aligns

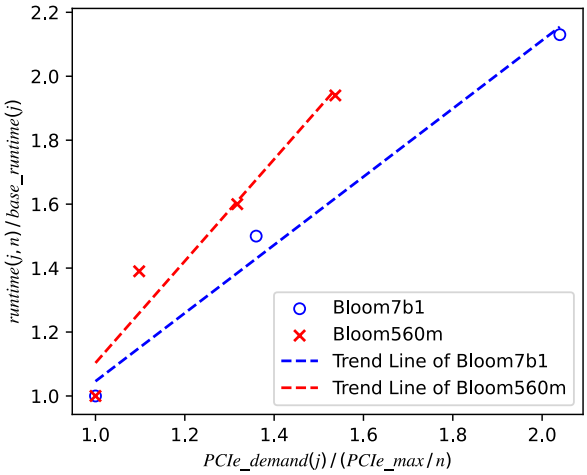


Figure 4: Calculate α_j using linear regression on the observed runtimes against the allocated PCIe bandwidth. The y-axis represents the job execution time slowdown. It is observed that a linear trend exists between slowdown and allocated PCIe bandwidth.

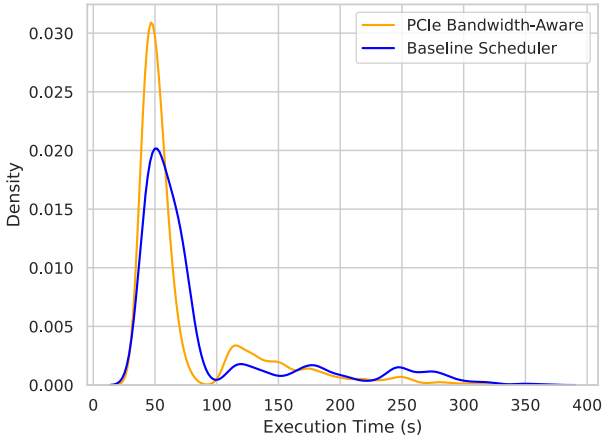


Figure 5: Kernel Density Estimate (KDE) plot of PCIe-bound job execution time on the testbed. Jobs scheduled by the PCIe bandwidth-aware scheduler generally have shorter execution times compared to those scheduled by the baseline scheduler.

with the experiment results from our testbed in Fig. 6. We gathered JCT data for each scheduler from both the real and simulation experiments, paired the results, and computed the correlation coefficient for each scheduler. The correlation coefficients for both the baseline and PCIe bandwidth-aware schedulers are 0.99, demonstrating the simulator's reliability.

Simulation of Larger-Scale Systems To evaluate the scalability of our scheduler in larger-scale systems, we use our simulator to model a system of 60 GPUs to process 1,400 jobs. Fig. 8 shows the

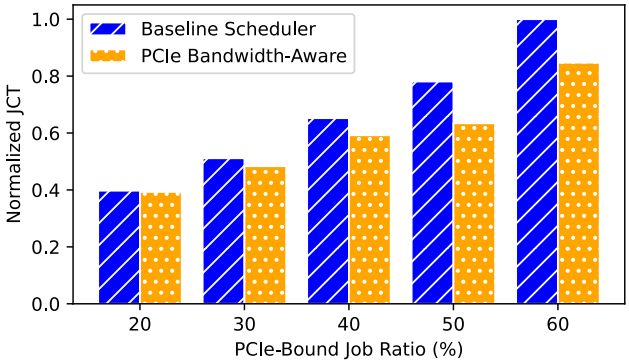


Figure 6: Job completion time (JCT) under different ratio of PCIe-bound job on the testbed with 4 A100 GPUs. The PCIe bandwidth-aware scheduler consistently outperforms the baseline scheduler.

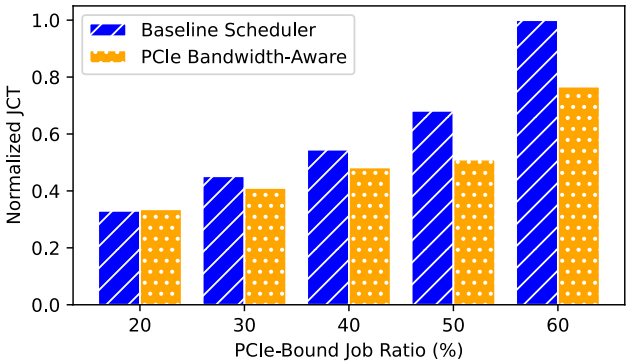


Figure 7: Normalized job completion time (JCT) under the simulation of 4 A100 GPUs. The results from the simulator closely align with the experimental outcomes obtained from the testbed.

job completion time for both our PCIe bandwidth-aware scheduler and the baseline scheduler. Notably, even within a cluster of 60 GPUs, our scheduler consistently outperforms the baseline. The reduction in JCT becomes more pronounced as the proportion of PCIe-bound jobs increases, with the aggregated job completion time decreasing by up to 17% when PCIe-bound jobs constitute 60% of the workload. These results from the larger-scale simulation align with our findings from the small-scale testbed, verifying that our scheduler achieves superior performance across various system scales.

Simulation of Various Maximum MIG Instances. In this experiment, we simulate GPUs capable of supporting a greater number of MIG instances to assess how these configurations influence the performance of our scheduler. Given the rapid growth in GPU computing power, it is reasonable to anticipate that future GPUs will be able to partition into more MIG instances than currently possible. For instance, current NVIDIA A100 GPUs can be divided into a maximum of seven `1g.5gb` MIG instances. We hypothesize that future GPUs will support a higher maximum number

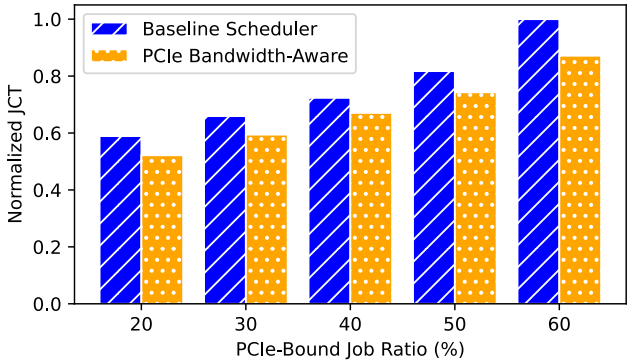


Figure 8: Normalized job completion time (JCT) under the simulation of a larger-scale system with 60 A100 GPUs.

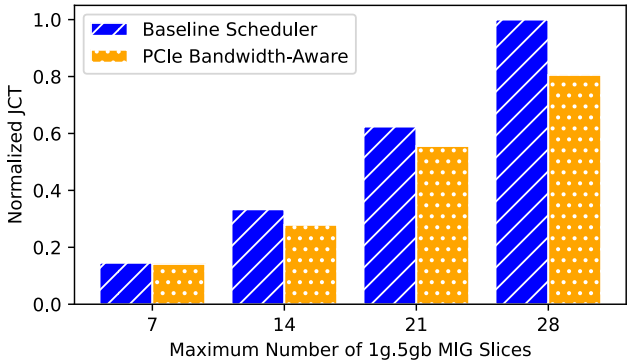


Figure 9: Normalized job completion time (JCT) under the simulation of GPUs with different maximum number of MIG instances.

of MIG instances. As shown in Fig. 9, an increased limit on the number of MIG instances results in more severe PCIe bandwidth contention. Consequently, the benefits of our scheduler become more evident, with the aggregated job completion time decreasing by up to 20%. This experiment highlights our scheduler's effectiveness in alleviating contention issues, particularly in scenarios where there is a significant imbalance between GPU computing power and PCIe bandwidth, which leads to more frequent PCIe bandwidth contention.

6 CONCLUSION

In this work, we introduced a PCIe bandwidth-aware scheduler designed to improve job scheduling efficiency in Multi-Instance GPU (MIG) environments by addressing PCIe bandwidth contention among concurrent instances on NVIDIA GPUs. Through a detailed analysis of PCIe bandwidth sharing behavior among MIG slices, we developed a performance model to predict slowdowns caused by PCIe bandwidth contention. By proactively scheduling jobs to minimize the overlap of PCIe-bound jobs, our approach achieved an approximate 18% reduction in aggregated job completion time compared to a baseline scheduler that does not consider PCIe bandwidth

contention. This reduction emphasizes the need of considering PCIe bandwidth as an important resource in multi-tenant GPU environments. Our findings demonstrate the potential of PCIe bandwidth-aware scheduling to enhance both efficiency and reliability in GPU resource sharing, particularly as GPU-based applications grow in complexity and diversity.

ACKNOWLEDGMENTS

The authors acknowledge the support from the National Science and Technology Council (NSTC) in Taiwan under grant numbers MOST 111-2221-E-007-064-MY3. The authors would also like to express their appreciation for the computational resources from IBM Thomas J. Watson Research Center used in this work.

REFERENCES

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[2] Mehmet E Belviranli, Farzad Khorasani, Laxmi N Bhuyan, and Rajiv Gupta. Cumas: Data transfer aware multi-application scheduling for shared gpus. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12, 2016.

[3] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[4] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers. *ACM SIGPLAN Notices*, 51(4):681–696, 2016.

[5] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. 2023.

[6] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. Characterizing multi-instance gpu for machine learning workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 724–731. IEEE, 2022.

[7] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 173–189, 2022.

[8] NVIDIA. Nvidia multi-instance gpu (mig) user guide. https://docs.nvidia.com/datacenter/tesla/pdf/NVIDIA_MIG_User_Guide.pdf, 2024.

[9] NVIDIA. Nvidia nsight systems. https://developer.nvidia.com/nsight-systems, 2024.

[10] Chris Porter, Chao Chen, and Santosh Pande. Compiler-assisted scheduling for multi-instance gpus. In *Proceedings of the 14th Workshop on General Purpose Processing Using GPU*, pages 1–6, 2022.

[11] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 322–337, 2019.

[12] Mark Van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse. Gpu-accelerated and parallelized elm ensembles for large-scale regression. *Neurocomputing*, 74(16):2430–2437, 2011.

[13] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. {MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, 2022.

[14] Zichao Yang, Heng Wu, Yuanjia Xu, Yuewen Wu, Hua Zhong, and Wenbo Zhang. Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous gpus. *IEEE Transactions on Computers*, 2023.

[15] Huaizheng Zhang, Yuanming Li, Wencong Xiao, Yizheng Huang, Xing Di, Jianxiong Yin, Simon See, Yong Luo, Chiew Tong Lau, and Yang You. Migperf: A comprehensive benchmark for deep learning training and inference workloads on multi-instance gpus. *arXiv preprint arXiv:2301.00407*, 2023.