Exploring the Diversity of Multiple Job Deployments over GPUs for Efficient Resource Sharing

Theodora Adufu*
Department of Computer Science
Sookmyung Women's University
Seoul, South Korea
theoadufu@sookmyung.ac.kr

Jiwon Ha†
Department of Computer Science
Seoul National University
Seoul, South Korea
jwh0245@naver.com

Yoonhee Kim‡
Department of Computer Science
Sookmyung Women's University
Seoul, South Korea
yulan@sookmyung.ac.kr

***Abstract*—Graphic Processing Units (GPUs) are gradually becoming mainstream computing resource for efficient execution of applications both on-premises and in the cloud. Currently however, most HPC applications are unable to leverage the large computing capabilities they provide leading to issues of resource under-utilization. Various GPU sharing approaches have been proposed which leverage either software or hardware level mechanisms like MPS or MIG in NVIDIA GPUs. However, combining both the software and hardware level technologies in an effort to mitigate resource under-utilization issues is yet to be fully explored. In this paper, we conduct a case study on scheduling memory intensive and compute intensive applications on an NVIDIA A30 GPU. We compare the performance when using only hardware level sharing mechanisms and when using both hardware and software level mechanisms. We observed that by combining both mechanisms, we improved total execution times by up to 14% for a single run whilst improving peak bandwidth utilization by about 39% for SCAN application.**

***Index Terms*—Spatial Sharing, Concurrency, Resource under-utilization, MPS, MIG**

I. Introduction

Cloud computing, the on-demand delivery of IT resources over the Internet [1], has changed the way scientists, businesses and individuals deploy applications. Traditionally, on-premises supercomputers were instrumental in deploying applications like HPC applications however, these were very expensive and required highly skilled personnel to manage them thus deterring many businesses from using computing resources [2] [3]. With cloud environments like [4]–[7], however, more and more users now have access to computing devices, databases and storage on-demand at a relatively low cost based on the pay-as-you-go pricing model most of these cloud providers use.

With the thousands of processor cores they provide, General Purpose Graphics Processing Units (GPGPU) have become one of the necessary computing resources adopted by cloud service providers to meet user's increasing demand for computing resources. Cloud-based GPUs promise to offer higher computational capacity on-demand which would further lower the costs of deploying applications and services.

In reality however, dedicated GPUs seem to be more expensive as traditional GPU run-time environments allow for the execution of only one application at a time. Cloud providers are leveraging container orchestrators like Kubernetes [8] or Nomad [9] to schedule containerized applications unto the GPU, however, this requires complex configurations since the container orchestrators do not support the GPU by default [10]. Additionally, GPUs cannot be shared across containers and pods thus containers are granted access to an entire GPU or to several GPUs [11]. Moreover, researchers [13]–[21] have observed that most applications are unable to saturate the GPU resources dedicated to them resulting in resource under-utilization thus higher overall infrastructural costs.

On NVIDIA GPUs, maximizing concurrency through mechanisms like CUDA Streams [22] [23], Hyper-Q, Multi-Process Service (MPS) [24] as well as Multi-Instance GPU (MIG) [25] have been introduced to further improve utilization of GPU resources. These however fall short in ensuring GPU saturation since that is application dependent.

This paper seeks to investigate optimal application-aware approaches to scheduling HPC applications on an NVIDIA A30 GPU in a bid to maximize GPU resource utilization. Through this research we

- investigate the utilization of different GPU resources when HPC applications are deployed on heterogeneous GPU resources
- investigate the configuration of MIG instances which best suits the execution of HPC applications with different resource demands
- maximize the resource utilization of GPU resources by exploring the use of a combination of software and hardware level spatial sharing mechanisms.

The rest of the paper is organized as follows: in Section 2, we briefly discuss the motivation for this investigation, some GPU sharing mechanisms and the issue of resource under-utilization. In Section 3, we discuss some related works on GPU resource sharing mechanisms and explain our proposed

‡Corresponding Author: Sookmyung Women's University, Department of Computer Science, yulan@sookmyung.ac.kr

approach in Section 4. We evaluate our experiments in Section 5 and conclude the paper in Section 6.

II. Background and Motivation

A. GPU Resource Sharing

CUDA's programming model enables both temporal and spatial sharing of GPU resources through technologies like CUDA Streams [22] [23], Hyper-Q, Multi-Process Service (MPS) [24] and Multi-Instance GPU (MIG) [25]. While CUDA streams facilitates concurrency among processes within a particular application through temporal sharing [33] of GPU resources, MPS and MIG enable spatially share GPU resources among concurrent applications.

Multi-Process Service(MPS) [24] allows spatial sharing of GPU resources across applications based on each application's resource(e.g. SM, memory, etc) demands. The software-based mechanism, allows users to specify the percentage of SMs to be used by a particular process while other resources like memory, memory bandwidth and caches are shared between concurrent applications.

Multi-Instance GPU(MIG) [25] is a hardware level sharing mechanism that partitions GPU resources into multiple predefined and isolated GPU Instances (GIs) [26] with dedicated compute cores(GPU SM slices), memory (GPU Memory slices), L2 cache and memory bandwidths to enable concurrent executions while providing isolation. MIG instances can be created from a combination of the aforementioned partitioned resources to the tune of the available resources. On the NVIDIA A100 [25] for instance, there are 19 possible MIG configurations into which the GPU can be partitioned statically and by default, each MIG instance allows for the execution of only one application at a time.

B. GPU Resource Under-utilization

Following research works [13]–[16] which investigated resource utilization of Deep Learning (DL) applications on GPUs, we presented in a previous paper [27], our observations of the utilization of SM and memory bandwidth resources when HPC applications are executed in isolation on a whole GPU and on differently-sized MIG instances (Figure 1 and Figure 2).

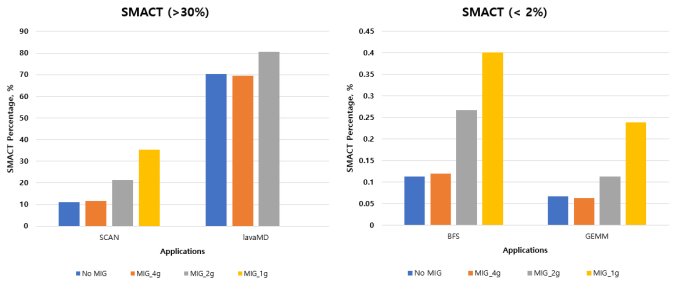

Fig. 1. SMACT [27]

We [27] measured amongst other metrics, the the ratio of cycles an SM has at least one warp active on a multiprocessor averaged over all SMs (SMACT) and the utilization

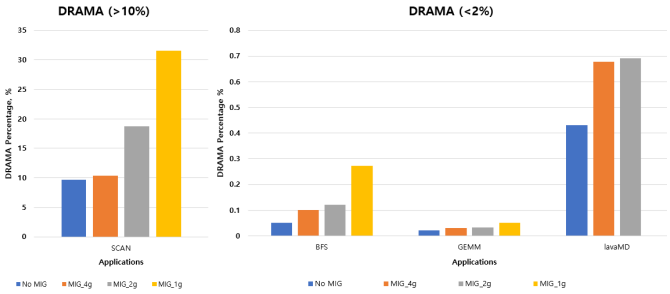

Fig. 2. DRAMA [27]

of the memory bandwidth (DRAMA) for 4 selected HPC applications taken from CUDA samples [28], Rodinia [29] and PolyBench [30] benchmarks using NVIDIA's Data Center GPU Manager (DCGM) [31]. We also compared the utilization obtained using DCGM to the compute and memory throughput (Figure 3) obtained using Nsight Compute [32].

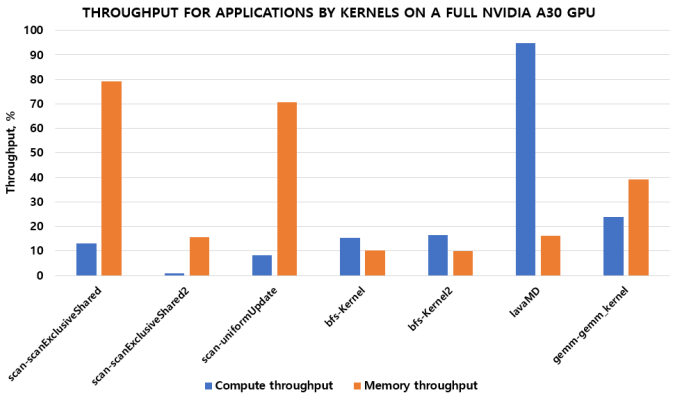

Fig. 3. Compute and Memory Throughput for Applications Executed on NVIDIA A30 GPU (Partially referenced from [27])

From Figure 1 we observed that, apart from LavaMD, the reported SM Activity for the applications investigated were relatively small (less than 10%) regardless of the size of MIG instance on which they were deployed. This revealed that the SM resources available to the applications were poorly utilized throughout the execution cycle. The observation also corresponded to the compute and memory throughput shown in Figure 3.

We also observed that from Figure 2 and Figure 3, SCAN used the device memory interface more actively than LavaMD, BFS and GEMM since the operations in SCAN require continuous reading and writing from the GPU memory and can thus be considered a memory intensive application. However, this was also insufficient to fully saturate the memory bandwidth of the NVIDIA A30 GPU. From the aforementioned observations, we safely concluded that majority of HPC benchmark applications under-utilize even the least-sized MIG instance.

III. RELATED WORKS

Until now, research has focused on improving the resource utilization of SMs either at the software level through technologies like CUDA Streams, Hyper-Q and MPS or using hardware level technologies like MIG. GSLICE [16] for instance, employs spatial sharing in GPUs with MPS. Dhakal et al. observed that the performance of inference models improved until a point of diminishing returns (kneepoint) when the share of GPU resources allocated is continuously increased. Resources are then partitioned according to the knee-points of the co-located inference workloads to maximize resource utilization.

Similarly, Choi et al. [33], propose a new abstraction for GPUs called gpu-let, which can create multiple virtual GPUs out of a single physical GPU with spatial partitioning. They leveraged both temporal and spatial sharing mechanisms and proposed a scheme to allocate GPU resources to jobs in a manner that minimizes interference while maximizing the resource utilization and meeting Service Level Objectives (SLOs). The proposed solution however tackles the issue of resource utilization only at the software level without considering sharing at the hardware level.

Both of the aforementioned research works focus on DL and ML applications and do not consider applications of different characteristics from other domains. Porter et al. [34], developed a workload manager that combines compiler-based instrumentation with a run-time system that leverages MPS for HPC benchmark applications from the Rodinia benchmark suite. Their proposed framework, supports mixed jobs with isolation requirements as well as jobs that can share a GPU instance. From their experiments, relaxing isolation for some jobs running on MIG-enabled devices can improve throughput, job turnaround time, and memory utilization, without heavily affecting kernel execution times.

Renowned schedulers like Slurm [35], treat NVIDIA Multi-Instance GPU (MIG) instances as individual GPUs. According to [34] however, because Slurm reserves a partition that is able to hold the process' maximum kernel size for the entire process' lifetime, there is GPU under-utilization in the schedules generated by Slurm, as well as sub-optimality in terms of a schedule's objective function such as throughput or the job turnaround time of a batch. Moreover, none of these prior research works leverage both MPS and MIG technologies during execution.

IV. MAXIMIZING RESOURCE UTILIZATION USING MPS IN MIG INSTANCES

To maximize resource utilization on the GPU as well as maximize the infrastructural costs of deploying applications over cloud GPU resources, there is the need for a scheduling framework that facilitates the sharing of the GPU at both the software and hardware levels. Leveraging both hardware and software level spatial sharing technologies promises to further improve GPU utilization as shown in Figure 4 whilst improving the overall performance of the GPU.

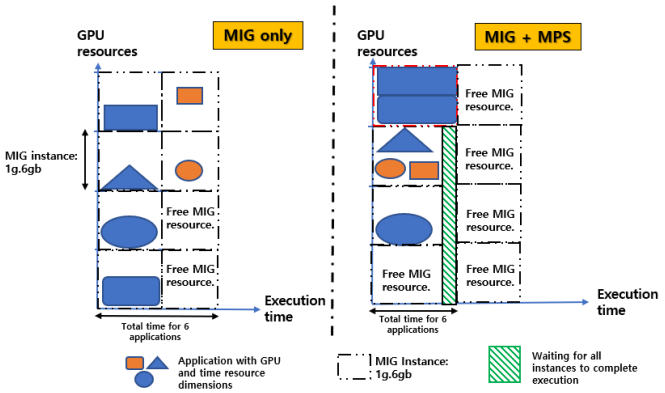

Fig. 4. Improvement in resource utilization and overall performance

MIG partitions a single GPU into multiple predefined and isolated GPU Instances (GIs) [26] on which workloads can be executed in parallel without the bottlenecks of error and security related interference. However, the traditional MIG spatial sharing mechanism allows for the deployment of only one application at a time in an instance. Consequently, when the resource utilization of an application deployed within an MIG instance is low and that application has a low SM occupancy (please refer to Figure 7 in previous work [27]), this only leads to issues like resource fragmentation described in [36]. When concurrently running applications with low SM occupancy using MPS [24] however, the MPS server balances workloads by over-lapping kernel and memcopy operations of the different processes running on the GPU and eliminating the overhead of context switching.

Leveraging both MPS and MIG spatial sharing technologies during execution, enables all shared applications to

- saturate the whole GPU as they saturate each MIG instance (Figure 4).
- improve the utilization of the GPU without compromising the overall performance of all scheduled applications.
- mitigate the issue of resource fragmentation [36] by providing sufficient isolated resources to applications with low resource use while freeing unused resources for use by other applications. This is especially useful in large clusters which require strong-scaling.

Li et al. [37] propose a technique that requires the use of both MPS and MIG when spatially sharing GPU resources. They first determine the amount of SM resources used when applications share GPU resources using MPS and then using a performance predictor, they estimate the optimal MIG partition required for the execution of a mix of jobs. For their research, the information obtained using MPS is only used to determine the optimum partition of GPU resources. They do not explore the use of both approaches during the actual execution of the applications.

Our investigations on the other hand, goes beyond using MPS to estimate the MIG partition to actually leveraging both technologies during execution. We also evaluate HPC

applications which have different characteristics.

V. Experiment and Evaluation

We conducted our investigations using two selected applications on the NVIDIA A30 GPU (Table I) briefly described below:

SCAN [28] is a simple parallel algorithm based on the all-prefix-sums operation and used in sorting, lexical analysis, string comparison, polynomial evaluation, stream compaction, building histograms and data structures (graphs, trees, etc.) amidst others [38]. It is classified as a memory intensive application (Figure 2 and Figure 3) in this study.

LavaMD [29], a molecular dynamics application that calculates the potential and relocation of particles within a large 3D space is classified as a compute intensive application (Figure 2 and Figure 3) in this study.

TABLE I
Experimental Set-up

GPU Device	NVIDIA A30
Device Memory	24GB
GPU memory bandwidth	933 GB/s
CUDA version	12.0
Nvidia-smi/ GPU Drivers	525.60.13
DCGM version	3.1.3
Nsight Compute version	2022.04

On the NVIDIA A30 GPU, there are 3 possible GPU Instances (GIs) which can be created (Table II) and which are used in our experiments. We create the instances in advance and do not take into account the time taken to create the instances during our investigations.

TABLE II
Resources for MIG instances on the A30 GPU

Profile name	Compute (GPC)	SM	Memory (GB)	L2 Cache (MB)
MIG 4g.24gb	4	56	24	24
MIG 2g.12gb	2	28	12	12
MIG 1g.6gb	1	14	6	6

A. CASE STUDY: Scenarios for Co-Running HPC Applications in MIG Instances

For our evaluation, we conducted a case study for co-running applications on the NVIDIA A30 GPU using MPS in differently configured MIG instances. Figure 5 shows the optimal execution configuration for co-running applications with different minimum resource requirements. We recall that, on the NVIDIA A30 GPU, there are five (5) possible MIG configurations as shown in Figure 5. We considered 5 possible cases where a user submitted a batch of 3 applications (2 SCAN and 1 LavaMD) to be executed concurrently on various MIG instances on NVIDIA A30 GPU as shown and evaluated these cases using both software and hardware level sharing approaches.

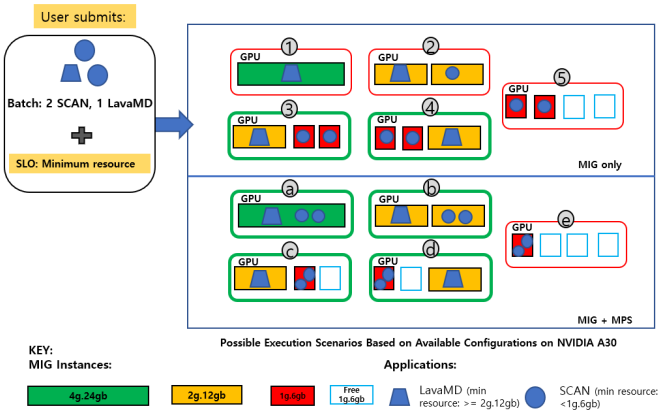

Fig. 5. CASE STUDY: Possible Execution Scenarios Based on Available Configurations on NVIDIA A30

From our previous research ([27]), we profiled both SCAN and LavaMD on different MIG instances and obtained information on the minimum resource required to run both applications; less than MIG_1g.6gb and a minimum of MIG_2g.12gb respectively. We thus performed our investigations, providing these resource requirements as part of the user-submitted Service Level Objectives (SLOs) in this study. We run the submitted applications for each of the cases **(MIG only: 1-5 and MIG + MPS: a-e)** presented in Figure 5 and evaluated our observations.

B. Evaluation: Optimum MIG Configuration

From Figure 5, we observed that, for a batch of jobs containing two (2) applications of SCAN and one (1) application of LavaMD as shown, not all MIG configurations can allow for the execution of the jobs in a single run. The configurations which can allow for the execution of all three applications concurrently is the heterogeneous configuration of either Scenario **3** or Scenario **4** when only MIG is used, and Scenarios a-d when MPS is used within the MIG instances. Scenarios **1** and **2** require at least 2 runs to ensure that all the applications are executed which could translate into higher costs of provisioning an MIG instance in cloud environments. Using MPS in MIG however enables users to save on these costs through concurrent executions within instances. Scenarios **5** and **e** however, would not allow for the execution of LavaMD since the minimum resource requirement for LavaMD (MIG_2g.12gb instance) cannot be satisfied. This suggests that MPS in MIG would be more efficient in heterogeneous resource environments.

C. Evaluation: Improvement in Execution time

Figure 6 shows the total execution times when the submitted applications are run using each GPU sharing approach for the 5 cases in Figure 5. The naming convention **Scenario 1_a**, used to make the graph more readable, shows the execution times when run using only MIG on the left and when using both MPS and MIG on the right for each entry. From Figure 6, we observed with reference to **Scenario 1_a** that, by using

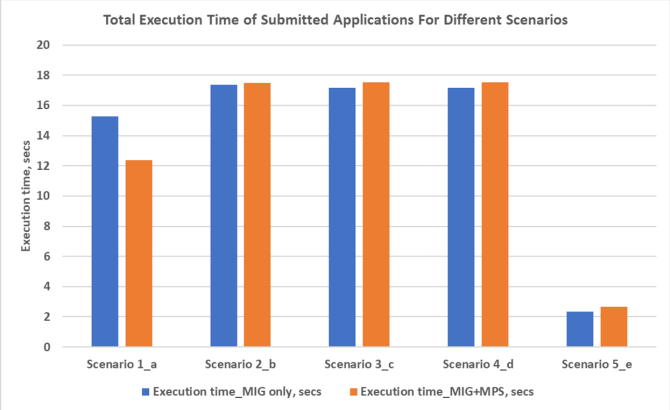

Fig. 6. Total Execution Time of Applications

MPS in MIG instances, the total execution times can be reduced by up to 14% when applications are concurrently run in MIG instances using MPS. This is particularly important in cases where the workloads are diverse in the demand for resources. Allowing jobs with low resource demands to run together with jobs high resource demands through MPS would allow the MPS server to over-lap kernel and memcopy operations thus reducing the total execution time. Also, by using MPS in MIG as in Scenario **a**, all user submitted applications are executed in a single run as opposed to three runs in Scenario **1**. **Scenario 2_b** shows that both applications complete execution in about the same time since for Scenario **2** the second SCAN application is executed concurrently with the execution of LavaMD after the first one is completed. From our investigations the worst case scenarios involve running LavaMD concurrently with other applications in different MIG instances as shown (Figure 6). We attribute this to a contention for CPU resources by each MIG instance.

D. Evaluation: Improvement in Resource Availability

Figure 7 shows the number of applications executed concurrently in a single run for each case as well as the total number or MIG instances freed in each execution run for each of the cases in Figure 5.

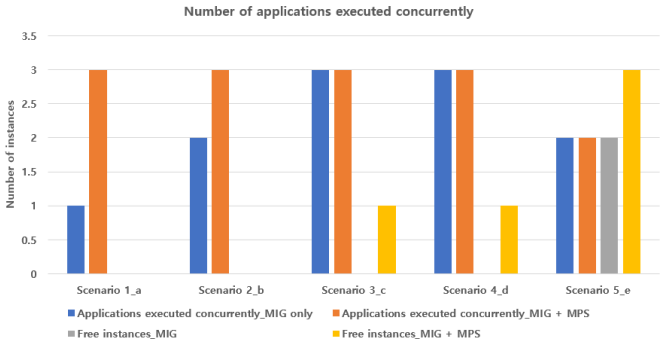

Fig. 7. Number of Concurrently Executed Applications with Corresponding Number of Freed MIG Instances

As explained previously, **Scenario 1_a** for instance, shows the execution times when run using only MIG on the left and when using both MPS and MIG on the right. From Figure 7, we observed with reference to **Scenario 3_c** and **Scenario 4_d** that, by using MPS in MIG instances, MIG instances can be freed for use by other applications. **Scenario 5_e** shows free MIG instances when both sharing mechanisms are used. This however is because, with the given configuration, the minimum resource requirement of LavaMD is not met and thus cannot be executed on the GPU. This reveals for further research, the need for application-aware scheduling policies to minimize failed jobs in large clusters. It also highlights the need to accurately configure the GPU to maximize the benefits of using both MIG and MPS sharing mechanisms.

E. Evaluation: Improvement in resource utilization

Figure 8 shows the SM resource utilization of the applications executed for Scenarios **1** and **a** in Figure 5. From Figure 8, we observed that **SMACT_a** corresponding to **Scenario a**, is higher especially during the initial stages of the execution. This is because the SCAN applications begin executions on the GPU before LavaMD application and thus MPS is able to overlap kernel operations between the SCAN applications during that period. LavaMD after completion of the CPU related tasks then begins executions on the GPU with a high resource use of 1. For Scenario **1** on the other hand, the sequential executions of the applications is seen to result in lower resource utilization (about 39% lower) especially when SCAN applications are being executed on the GPU. The utilization remains high at 1 for both scenarios during the execution of LavaMD.

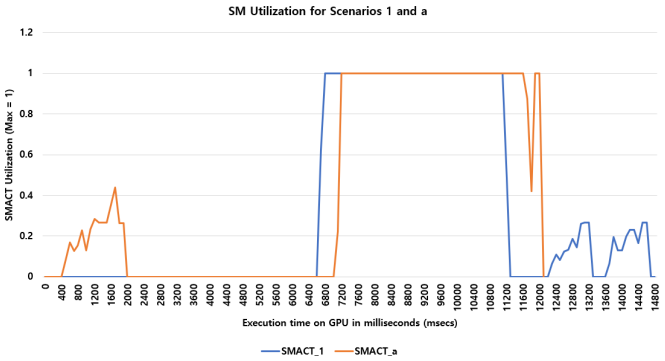

Fig. 8. Improvement in Resource Utilization using MPS in MIG instances

VI. Conclusion and Future Works

This study seeks to investigate the effect of leveraging both software and hardware level GPU spatial sharing mechanisms on performance and the freeing up of resources whilst improving resource utilization. From our study, we observed that, by combining both software and hardware level GPU-sharing mechanisms, a user is able to improve overall performance, reduce the number of execution runs and free-up MIG instances or GPU resources for use by other applications for most cases.

In the future we intend to investigate various application-aware scheduling policies using a larger mix of HPC applications.

VII. Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.2021R1A2C1003379).

References

[1] Cloud computing, https://aws.amazon.com/what-is-cloud-computing/
[2] HPC in the cloud, https://www.ni-sp.com/support/hpc-in-the-cloud/
[3] 2022 State of Computational Engineering Report, https://rescale.com/resources/2022-state-of-computational-engineering-report/
[4] IBM Cloud Server's NVIDIA GPU, https://www.ibm.com/kr-ko/cloud/gpu?mhsrc=ibmsearch_a&mhq=GPU, (Last accessed, June 27, 2023)
[5] Recommended GPU instances, https://docs.aws.amazon.com/ko_kr/dlami/latest/devguide/gpu.html, (Last accessed, June 27, 2023)
[6] Cloud GPU, https://cloud.google.com/gpu?hl=ko, (Last accessed, June 27, 2023)
[7] Elastic GPU, https://www.alibabacloud.com/ko/product/gpu, (Last accessed, June 27, 2023)
[8] About GPUs in GKE, https://cloud.google.com/kubernetes-engine/docs/concepts/gpus
[9] Nomad, https://www.nomadproject.io/
[10] GPU Scheduling: what are the options?, https://www.run.ai/guides/multi-gpu/gpu-scheduling
[11] https://www.run.ai/guides/multi-gpu/gpu-scheduling
[12] GPU Cloud Computing, https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/ (Last accessed, June 27, 2023)
[13] Dhakal A, Cho J, Kulkarni SG, Ramakrishnan KK, Sharma P., Spatial Sharing of GPU for Autotuning DNN models. arXiv preprint arXiv:2008.03602. 2020 Aug 8.
[14] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, Xiang Chen., A Survey of Multi-Tenant Deep Learning Inference on GPU. [Online].
[15] Fuxun Yu, Zirui Xu, Tong Shen, Dimitrios Stamoulis, Longfei Shangguan, Di Wang, Rishi Madhok, Chunshui Zhao, Xin Li, Nikolaos Karianakis, et al. "Towards latency-aware DNN optimization with GPU runtime analysis and tail effect elimination." arXiv preprint, 2020. [Online]. Available: arXiv:2011.03897.
[16] Aditya Dhakal, Sameer G Kulkarni, K. K. Ramakrishnan. "GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform." [Online]. Available: [URL]
[17] Aggelos Ferikoglou and Dimosthenis Masouros and Achilleas Tzenetopoulos and Sotirios Xydis and Dimitrios J. Soudris, Resource Aware GPU Scheduling in Kubernetes Infrastructure, PARMA-DITAM@HiPEAC, 2021
[18] Min-Chi Chiang and Jerry Chou, DynamoML: Dynamic Resource Management Operators for Machine Learning Workloads,In CLOSER, 2021
[19] Gingfung Yeung, Damian Borowiec, Adrian Friday, Richard Harper, and Peter Garraghan,"Towards GPU Utilization Prediction for Cloud Deep Learning", USENIX Workshop on Hot Topics in Cloud Computing, 2020
[20] , Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan, "Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems", IEEE Transactions on Parallel and Distributed Systems, 2022.
[21] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, et al. "fTVMg: An automated end-to-end optimizing compiler for deep learning." In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2018, pp. 578-594.
[22] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (Last accessed, June 30, 2023).
[23] https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/
[24] NVIDIA, https://docs.nvidia.com/deploy/mps/index.html(2020), (Last accessed, June 27, 2023).
[25] NVIDIA Multi-Instance GPUs, https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html (Last accessed, June 27, 2023).
[26] NVIDIA MIG User Guide, https://docs.nvidia.com/datacenter/tesla/pdf-/NVIDIA_MIG_User_Guide.pdf, (Last accessed, June 27, 2023).
[27] Adufu Theodora, Ha Jiwon, Kim Yoonhee, "An Analysis of Efficient GPU Resource Sharing for Concurrent HPC Application Executions", KNOMS Review, Vol.25, No. 1, September, 2022
[28] CUDA SAMPLES, https://github.com/NVIDIA/cudasamples/ (Last accessed, June 27, 2023)
[29] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," 2009 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA, 2009, pp. 44-54, doi: 10.1109/IISWC.2009.5306797.
[30] Louis-Noël Pouchet. 2012. Polybench: The polyhedral benchmark suite. http://www. cs. ucla. edu/pouchet/software/polybench. (Last accessed, June 27, 2023)
[31] NVIDIA, Data Center GPU Manager (DCGM) 3.1, https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/feature-overview.html (Last accessed, June 30, 2023)
[32] Nsight Compute, https://developer.nvidia.com/nsight-compute (Last accessed, June 27, 2023).
[33] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, Jaehyuk Huh. "Multi-model Machine Learning Inference Serving with GPU Spatial Partitioning."
[34] Chris Porter, Chao Chen, Santosh Pande, Compiler-Assisted Scheduling for Multi-Instance GPUs, Proceedings of the 14th Workshop on General Purpose Processing Using GPU, GPGPU '22 vol 4, pg 1-6, 2022. https://doi.org/10.1145/3530390.3532734.
[35] Slurm MIG, https://slurm.schedmd.com/gres.html.2021
[36] Qizhen Weng and Lingyun Yang and Yinghao Yu and Wei Wang and Xiaochuan Tang and Guodong Yang and Liping Zhang, Beware of Fragmentation: Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent, 2023 USENIX Annual Technical Conference (USENIX ATC 23), 2023, ISBN:978-1-939133-35-9, Boston, MA, 995–1008, https://www.usenix.org/conference/atc23/presentation/weng, USENIX Association
[37] Baolin Li and Tirthak Patel and Siddharth Samsi and Vijay N. Gadepally and Devesh Tiwari, "MISO: exploiting multi-instance GPU capability on multi-tenant GPU clusters", Proceedings of the 13th Symposium on Cloud Computing, 2022
[38] SCAN, https://www.eecs.umich.edu/courses/eecs570/hw/parprefix.pdf