https://doi.org/10.22670/knom.2023.26.1.11

HPC 응용들의 효율적인 동시 실행을 위한 GPU 자원 공유 방안 분석

테오도라 아두푸◆, 하 지 원*, 김 윤 희°

An Analysis of Efficient GPU Resource Sharing for Concurrent HPC Application Executions

Theodora Adufu◆, Jiwon Ha*, Yoonhee Kim°

요 약

그래픽 처리 장치(GPU)는 사용자에게 더 낮은 인프라 비용으로 다양한 애플리케이션의 실행을 가속화할 수 있는 기하급수적인 컴퓨팅 용량을 제공함에 따라 오늘날의 HPC 및 클라우드 인프라의 중요한 부분이 되었다. 그러나 기존 GPU는 한 번에 하나의 애플리케이션만 실행할 수 있으므로 리소스 활용도가 낮고 배치 비용이 증가하는 문제가 발생한다. 지금까지는 GPU에서 동시 실행을 가능하게 하기 위한 노력이 소프트웨어 기반이었으나, NVIDIA의 MIG 기능이 도입되면서 GPU를 분할하여 애플리케이션에 격리된 리소스를 제공할 수 있게 되었다. 본 논문에서는 다양한 MIG 인스턴스에서 실행되는 HPC 애플리케이션의 동작을 연구한다. 또한 하드웨어 수준에서 GPU 리소스를 공유하는 것이 각 애플리케이션의 성능에 미치는 영향에 대해서도 연구한다.

Key Words : Spatial Sharing, MIG Instances, Resource-utilization

ABSTRACT

Graphics Processing Units (GPU) have become a significant part of today's HPC and cloud infrastructures as they offer users exponential computing capacity to accelerate the execution of diverse applications at lower infrastructural costs. The traditional GPU however, allows for the execution of only one application at a time, giving rise to issues of resource under-utilization and subsequently higher deployment costs. Until now, efforts to enable concurrent executions on the GPU have been software-driven however with the introduction of the MIG feature by NVIDIA, GPUs can now be partitioned to provide isolated resources to applications. This paper investigates the behavior of HPC applications executed on different MIG instances. We also explore the implications of sharing GPU resources at the hardware level, on the performance of each application.

※ This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2021R1A2C1003379)
◆ First Author : Sookmyung Women's University, Department of Computer Science, theoadufu@sookmyung.ac.kr
° Corresponding Author : Sookmyung Women's University, Department of Computer Science, yulan@sookmyung.ac.kr
* Korea University, Department of Computer Science, jwh0245@naver.com

논문번호 : KNOM2023-01-05, Received July 2, 2023; Revised August 10, 2023; Accepted August 20, 2023

Ⅰ. Introduction

Scientists researchers and engineers over the years, have deployed various computing infrastructures to model new products and simulate the interactions of these products with the world. The provisioning of cloud platforms for instance, began a gradual shift from the use of traditional on-premises supercomputers to deploying applications including HPC applications in the cloud [1] [2]. A typical HPC application places significant demands on the compute hardware and infrastructure that supports it and hence may require the computing capabilities available through accelerators like the GPU. In cloud environments [3] [4] [5] [6], GPUs remain a necessary computing resource for the efficient execution of diverse workloads. Modern GPU architectures offer users exponential GPU capacity to accelerate the execution of diverse applications such as High Performance Computing (HPC) applications and Machine Learning (ML) applications. NVIDIA's GPU Cloud for instance offers researchers the flexibility to run HPC application containers on NVIDIA GPU's[7]. The opportunity to fuse HPC with Artificial Intelligence (AI) and ML is also fueling the advancement of computational science across a broad range of industries and domains [8].

Additionally, deploying HPC applications in cloud environments on a short-term basis enables scientists to quickly deploy applications at a relatively lower cost whilst leveraging the capabilities of better computing infrastructure. However in the medium to long run, this translates into higher costs especially for users of cloud environments since for the default GPU run-time environment, only one application can be executed at a time on the GPU.

Researchers [9] [10] [11] [12] also observed the issue of resource under-utilization in the default GPU run-time environment which has negative cost implications for the user. Using Deep Learning Inference Models, researchers [13] [14] [15] [16] [17] have investigated the possibility of saturating all Streaming Multiprocessors (SMs) of the GPU in a bid to improve GPU utilization. However, these reveal issues regarding optimum resource allocation, performance isolation, interference, and performance optimization.

A common approach to maximize resource utilization is to maximize the concurrency of applications executed on the GPU. NVIDIA's CUDA Streams [18] [19] for instance, allows applications to improve GPU utilization by interleaving operations in different streams within the application in order to achieve concurrency. NVIDIA has also introduced spatial sharing approaches like Hyper-Q and it's Multi-Process Service (MPS) [20] as well as Multi-Instance GPU (MIG) [21] to further improve resource utilization by allowing multiple contexts to be executed in parallel on shared GPU resources.

This paper seeks to investigate the behaviour of selected HPC workloads when deployed on various MIG instances. Through this research, we

- investigate the utilization of different GPU resources when applications are deployed on different MIG instances
- investigate the characteristics of selected HPC applications by their performance on various MIG instances.

Our investigations reveal that:

- Using smaller MIG instances has significant benefits including freeing computing resources for use by other applications without significant performance degradation.
- For compute intensive models, smaller MIG instances may not be sufficient for executions thus characterizing the application before execution is necessary for proper resource allocation.

The rest of the paper is organized as follows: in Section 2, we briefly discuss some related works on resource sharing approaches on GPUs and explain the motivation for this investigation through our experimental study in section 3. In Section 4, we give a description of GPU sharing using MIG and explain our experimental setup in

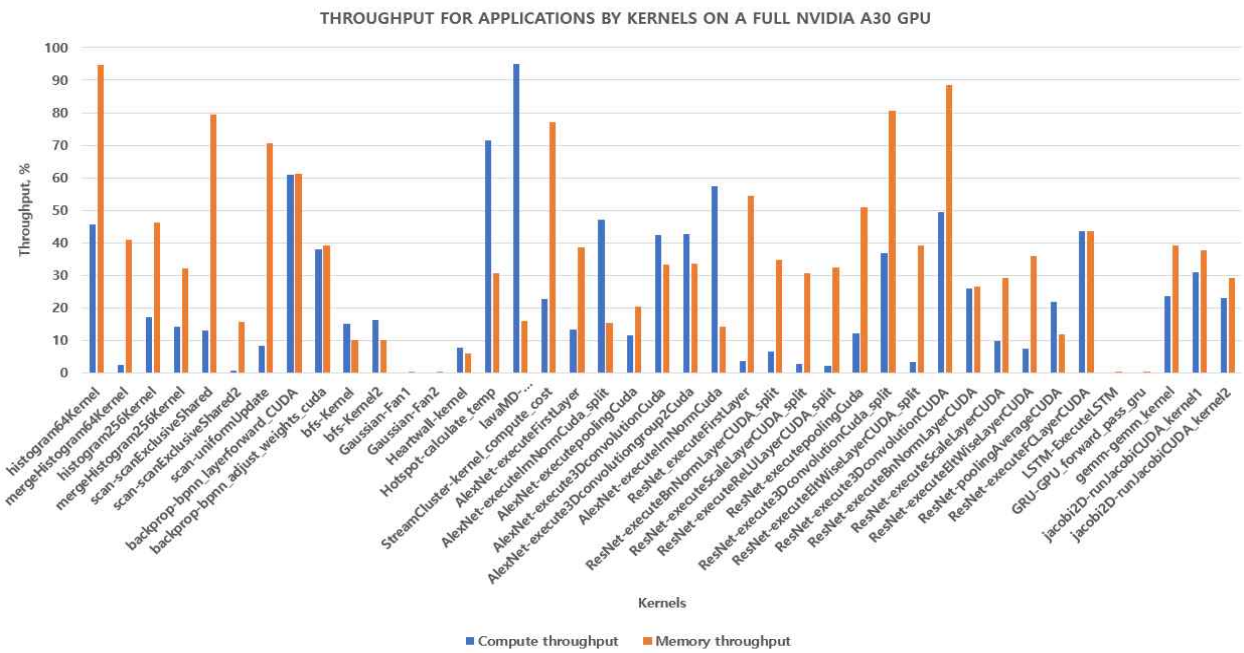


Fig. 1. Poor resource utilization of the NVIDIA A30 GPU

Section 5. We present the quantitative results of our investigations in Section 6 and conclude the paper in Section 7.

II. Related Works

A. Resource Sharing Approaches

Resource sharing approaches deployed on GPUs can be classified into either temporal sharing (time-slicing) or spatial sharing (hardware-based sharing). The CUDA programming model provides multiple technologies for concurrently executing applications on GPU resources. CUDA Streams for instance, allows processes within a particular application to be submitted to different independent queues for independent execution by the GPU. With CUDA streams, resources such as SMs, Memory, Memory bandwidth, caches are all shared by the processes in the various streams of an application in a time-sharing or temporal sharing [22] manner. Thus, although CUDA Streams tend to improve GPU utilization and overall throughput, they have an adverse impact on latency [12]. CUDA graphs [23] allow work to be defined as graphs rather than single operations to mitigate some of these overheads.

Multi-Process Service (MPS) [20] is a software-based mechanism which allows spatial sharing of GPU resources across multiple co-operative processes of CUDA kernels like MPI jobs via a hardware feature called Hyper-Q. By leveraging GPU's parallel compute capability, MPS shares the GPU's SM resources across applications based on their resource demands. This uncontrolled spatial sharing of GPU resources cannot guarantee performance isolation and often results in unpredictable application throughput and latency [24]. MPS alternatively allows users to determine a percentage limit of GPU resources (SMs) to be used by a particular process when the compute resource is allocated. However, other resources like memory, memory bandwidth and caches are all shared between co-executing applications.

III. Experimental Study on Resource Utilization by HPC Applications

Following observations by researchers [9] [10] [11] [12] which revealed that Deep Learning (DL) applications under-utilize GPU resources, we conducted a preliminary experiment (Figure 1), to investigate the compute and memory throughput

of 15 selected HPC applications taken from CUDA samples [25], Rodinia [26], PolyBench [27] and Tango [28] Benchmarks on a full NVIDIA A30 GPU using the Nsight Compute profiler [29]. This was to ascertain whether the claims by these researchers were true for HPC applications as well.

From our experimental study Figure 1, we observed that, most HPC applications had less than 50% compute and memory throughput indicating that the claims were true for HPC applications as well. We observed also that there were only few applications which fully engaged the compute resources during the execution life-cycle. The memory throughput was higher for kernels in Histogram, SCAN, Resnet which are applications known to be memory intensive.

Following these observations, we proceed to investigate the behaviour of some of these applications on different MIG instances.

IV. GPU-Sharing with MIG

Multi-Instance GPU (MIG) [21] is a new technology introduced by NVIDIA that can partition a GPU to better fit workloads that require less compute and memory resources than is available on a full GPU. The MIG feature on the Ampere micro-architecture partitions the GPU resources into dedicated compute cores, memory, cache and memory bandwidths (Figure 2).

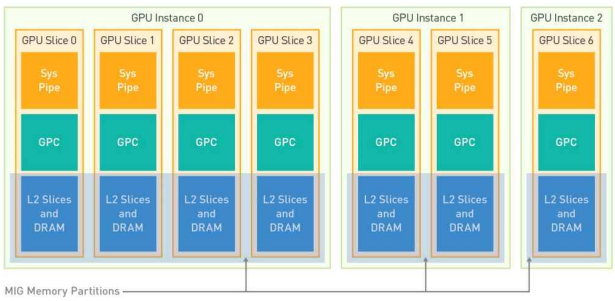


Fig. 2. MIG memory partitions for an MIG compute configuration with three GPU Instances on A100 [30]

Until now, research has focused on improving the resource utilization of SMs at the software level through technologies like CUDA Streams, Hyper-Q and MPS. However, this hardware-level partitioning technology serves as a solution to the problem of resource under-utilization as well as providing isolation by partitioning the GPU resources into dedicated compute cores, memory, cache and memory bandwidths known as a GPU Slice.

A GPU Slice [31] or GPC Slice is the smallest fraction of the GPU that combines a single GPU memory Slice and a single GPU SM Slice. A GPU Memory Slice, the smallest fraction of GPU memory, includes all the isolated blocks of L2 cache slices, associated frame buffer memory and proportions of memory bandwidth available for computation. Similarly, a GPU SM Slice is the smallest fraction of SMs on the GPU available for computation when configured in MIG mode.

MIG partitions a single GPU into multiple predefined and isolated GPU Instances (GIs) [32] on which workloads can be executed without the bottlenecks of error and security related interference. A GPU Instance (GI) is a combination of GPU slices and GPU engines which can be further partitioned into MIG instances. Within a GI, GPU Memory and GPU SM slices are shared and the SM slices can further subdivided into Compute Instances (CI). MIG instances can thus be created from a mix and match of GPC Slices so long as there are enough resources available to execute the workload.

On the NVIDIA A100 which serves as the representative GPU for the MIG feature [21], there are 19 possible MIG configurations as shown in Figure 3. A GPU can be re-partitioned into any of these configurations though not without limitations. For instance, since MIG re-partitioning is not dynamic on the A100 and A30 GPUs, there is the issue of poor resource allocation stemming from poor configurations. This could further lead to inefficiencies in GPU utilization and higher infrastructure costs.

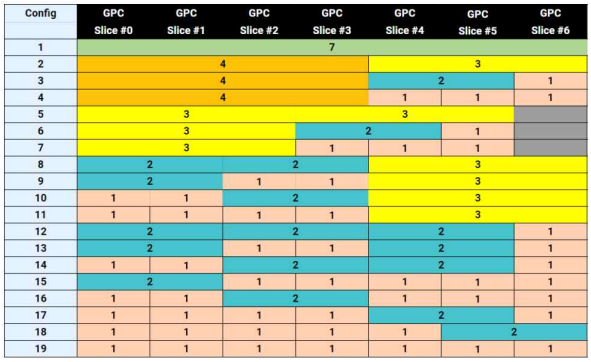


Fig. 3. Supported profiles of Multi Instance GPUs on NVIDIA A100 architecture [21]

V. Experiment Environment

We conduct our experiments on the NVIDIA A30 GPU (Table I) which allows for 5 possible MIG configurations [21] as shown in Figure 4.

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

For these 5 configurations, there are a total of 3 possible GPU Instances (GIs) which can be created on the A30 architecture (Table I) and which are used in our experiments.

Fig. 4. Supported profiles of Multi Instance GPUs on NVIDIA A30 architecture [21]

Table II shows the resources for each of the available GPU Instances. We create the instances in advance and do not take into account the time taken to create the instances during our investigations.

A. Applications/Workloads:

SCAN [25]: This 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 [33].

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

Breadth-First-Search (BFS) [26]: This is a graph traversal problem, which is commonly used to find the shortest path between two nodes.

LavaMD [26]: This is a molecular dynamics application that calculates the potential and relocation of particles within a large 3D space.

GEMM [27]: This is an algorithm that multiplies two input matrices to produce an output matrix by partitioning the output matrix into tiles, which are then assigned to thread blocks. It serves as a fundamental building block for many operations in neural networks.

B. Profiling Metrics:

To evaluate the effectiveness of the proposed GPU sharing technology, we collected metrics on the SM activity (SMACT), the memory bandwidth utilization (DRAMA), the GPU engine activity (GRACT) and the occupancy of the SMs (SMOCC) every 100ms using NVIDIA Data Center GPU Manager (DCGM) [341]. We show graphically, the moving average for each metric per application in section 5. The metrics collected are described as follows:

GRACT shows the fraction of time any portion of the graphics or compute engines were active. It shows the ratio of time the graphics/compute context is bound and the graphics pipe or compute pipe is busy.

SMACT measures the ratio of cycles an SM has at least one warp is active on a multiprocessor averaged over all SMs. It is used in tandem with the GRACT metric to confirm the accuracy of the metrics collected.

SMOCC measures the fraction of resident warps on a multi-processor, relative to the maximum number of concurrent warps supported on a multiprocessor.

DRAMA essentially measures how active the device memory interface is in sending or receiving data from the global memory.

VI. Evaluation

In this section, we evaluate GPU sharing for 4 selected applications taken from from CUDA samples [25], Rodinia [26], PolyBench [27] benchmarks using the profiling metrics defined in section 4. We share our observations on the performance of applications on the various MIG instances by comparing the 4 metrics obtained using the DCGM profiler relative to executions on a full GPU without MIG mode enabled.

A. Graphics Engine Activity (GRACT)

From Figure 5, we observed that, the reported Graphics Engine Activity for BFS and GEMM are relatively small (less than 10%) regardless of the various MIG instances on which they are deployed.

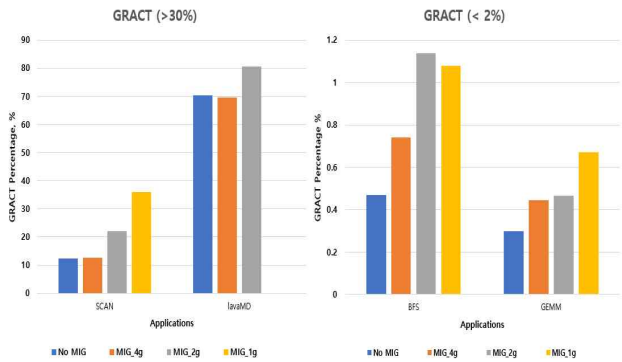


Fig. 5. GRACT

This revealed that BFS and GEMM were not directly affected by the availability of GPU resources as their utilization is relatively low. We also observed that the Graphics Engine Activity for the SCAN application was dependent on the available resources as the activity of the GPU was higher when SCAN was run on a MIG 1g.6gb instance compared to the MIG 2g.12gb instance (0.4x) and MIG 4g.24gb instance (2.8x). This revealed that for large applications implementing the SCAN algorithm, the size of the MIG instance must be carefully considered to optimize performance.

LavaMD application did not run on the MIG 1g.6gb instance since the available SMs were insufficient. This revealed the need to first characterize applications before allocating MIG resources to them as well as validated the isolation provided through the MIG technology.

B. Streaming Multiprocessor Activity (SMACT)

We also observed the SMACT for each application as shown in Figure 6. Similar to the observations for GRACT, we observed that LavaMD had the highest SM activity per SM showing that LavaMD is a compute intensive application since for more than 80% of the execution cycles, the assigned SMs have at least one warp assigned to them. BFS and GEMM show far less SM activity whilst the activity for SCAN varies across the MIG instances similar to observations for GRACT.

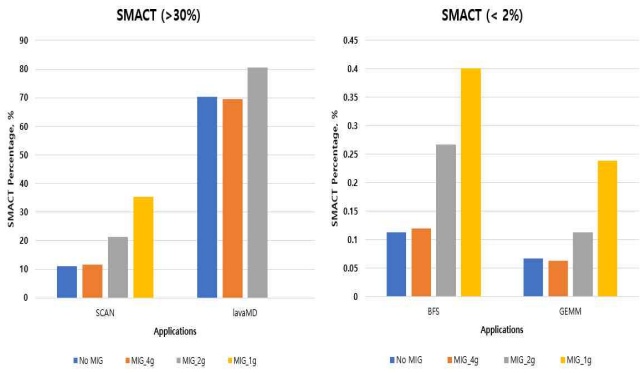


Fig. 6. SMACT

C. Streaming Multiprocessor and Occupancy (SMOCC)

From Figure 7, we observed that for LavaMD application, though the SMACT and GRACT recorded were high, the occupancy is relatively low. This revealed that there is the need to optimize the LavaMD application to maximize the fraction of resident warps on each SM during

execution. The SMOCC for SCAN, BFS and GEMM is consistent with the GRACT and SMACT showing that these applications have a good mapping of warps to SMs.

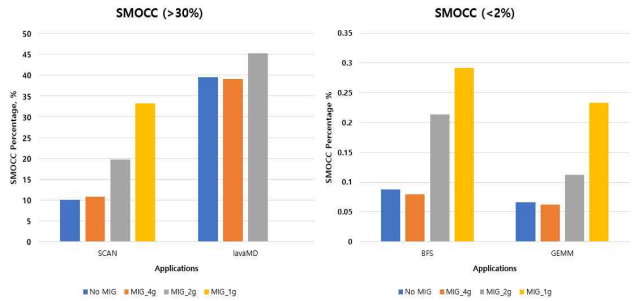


Fig. 7. SMOCC

D. DRAM Activity (DRAMA)

Figure 8 shows the DRAM activity for the applications. We observed that, SCAN uses 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.

Unlike the other applications, the size of the MIG instance significantly affects the DRAM activity recorded for SCAN. This revealed that, for memory intensive applications, the MIG instance allocated must have sufficient memory resources in order to mitigate possible bottlenecks.

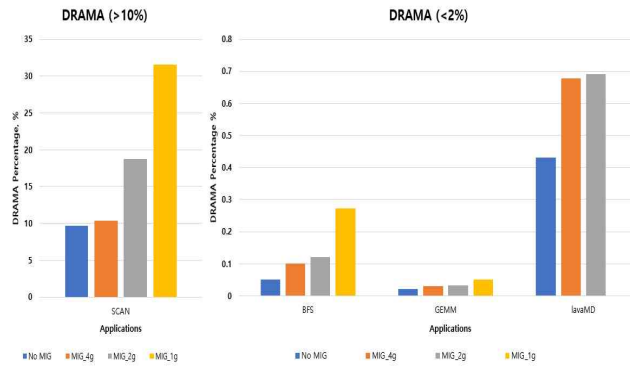


Fig. 8. DRAMA

E. Scenarios for Co-Running HPC applications in MIG instances

Having observed from our experiments that most applications are able to run successfully on smaller partitions of the GPU without significant overheads, we investigate scenarios for co-running applications on the NVIDIA A30 GPU. In Figure 9, we show the optimal execution configuration for co-running applications with different minimum resource requirements. We consider the case where a user submits a batch of 3 applications to be executed concurrently on MIG instances on NVIDIA A30 GPU.

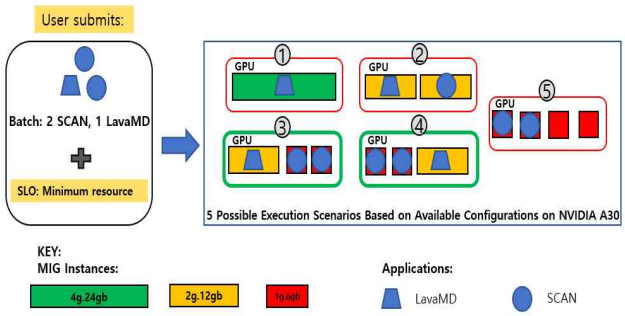


Fig. 9. 5 Possible Execution Scenarios Based on Available Configurations on NVIDIA A30

We recall that, on the NVIDIA A30 GPU, there are five (5) possible MIG configurations as shown in Figure 9. Thus if, for instance, the batch contains two (2) applications of SCAN and one (1) application of LavaMD, the MIG configurations which can allow for the execution of all three applications concurrently is either configuration 3 or configuration 4. This is due to the fact that, based on profiled information, the minimum resource required for the execution of LavaMD is the MIG 2g.12gb instance while the SCAN application can be executed on the MIG 1g.6gb instance. All other MIG configurations would require that at least one application waits for the completion of another application before being executed.

Consequently, for a case where the batch contains two (2) applications of LavaMD and one (1) application of SCAN, there will be no possible MIG configuration which can allow for the execution of all three applications concurrently since the minimum resource requirement for LavaMD is satisfied by the MIG 2g.12gb instance. In that case, the execution times of the applications may have to be considered as well.

Ⅶ. Conclusion and Future Works

The paper first confirms that when deploying HPC applications on full GPU resources, there is resource under-utilization. The paper then investigated the behaviour of selected HPC applications when deployed on various MIG instances. It was observed that, by deploying applications on smaller MIG instances, computing resources are freed for use by other applications without significant performance degradation. Applications can run thus be executed concurrently in different MIG instances without any interference as long as there are available resources.

From our investigations, we observed the need to first characterize applications before allocating MIG resources to them in order to determine the right MIG resource for execution. We also observed the for memory intensive applications, the MIG instance allocated must have sufficient memory resources in order to mitigate possible bottlenecks such as Out-Of-Memory (OOM) situations. We also recommend that HPC applications be optimized to maximize the fraction of resident warps on each SM during execution and hence improve resource utilization.

Though we do not take into account the time taken to create the instances in the experiments, we observed slight changes in the metrics collected on the full resource as compared to MIG 4g instance. For instance, for the LavaMD application, we observed a higher percentage of compute engine and SM activity seen in the values for GRACT and SMACT for a No MIG case and the MIG 4g case though the resources in each case are expected to be the same. We consider this as part of the overheads in deploying the applications in MIG instances and would investigate this further in future research works. We also intend to investigate various scheduling scenarios for MIG instances.

References

[1] HPC in the cloud, https://www.ni-sp.com/support/hpc-in-the-cloud/

[2] 2022 State of Computational Engineering Report, https://rescale.com/resources/2022-state-of-computational-engineering-report/

[3] IBM Cloud Server's NVIDIA GPU, https://www.ibm.com/kr-ko/cloud/gpu?mhsrc=ibmsearch a&mhq=GPU, (Last accessed, June 27, 2023)

[4] Recommended GPU instances, https://docs.aws.amazon.com/ko_kr/dl-ami/latest/devguide/gpu.html, (Last accessed, June 27, 2023)

[5] Cloud GPU, https://cloud.google.com/gpu?hl=ko, (Last accessed, June 27, 2023)

[6] Elastic GPU, https://www.alibabacloud.com/ko/product/gpu, (Last accessed, June 27, 2023)

[7] GPU Clouds, https://www.nvidia.com/en-gb/gpu-cloud/hpc-app-containers/ (Last accessed, June 27, 2023)

[8] GPU Cloud Computing, https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/ (Last accessed, June 27, 2023)

[9] Dhakal A, Cho J, Kulkarni SG, Ramakrishnan KK, Sharma P., Spatial Sharing of GPU for Autotuning DNN models. arXiv preprintarXiv:2008.03602. 2020 Aug 8.

[10] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, Xiang Chen., A Survey of Multi-Tenant Deep Learning Inference on GPU. [Online].

[11] Fuxun Yu, Zirui Xu, Tong Shen, Dimitrios Stamoulis, Longfei Shang-guan, 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.

[12] Aditya Dhakal, Sameer G Kulkarni, K. K.

Ramakrishnan. ”GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform.” [Online]. Available: [URL]

[13] Aggelos Ferikoglou and Dimosthenis Masouros and Achilleas Tzenetopoulos and Sotirios Xydis and Dimitrios J. Soudris, ResourceAware GPU Scheduling in Kubernetes Infrastructure, PARMADITAM@HiPEAC, 2021

[14] Min-Chi Chiang and Jerry Chou, DynamoML: Dynamic Resource Management Operators for Machine Learning Workloads,In CLOSER, 2021

[15] 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

[16] , 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.

[17] 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.

[18]https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (Last accessed, June 30, 2023).

[19]https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/

[20] NVIDIA, https://docs.nvidia.com/deploy/mps/index.html(2020), (Last accessed, June 27, 2023).

[21] NVIDIA Multi-Instance GPUs, https://docs.nvidia.com/datacenter/tesla/mig-

user-guide/index.html (Last accessed, June 27, 2023).

[22] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, Jaehyuk Huh. ”Multi-model Machine Learning Inference Serving with GPU Spatial Partitioning.”

[23] CUDA Graphs, https://developer.nvidia.com/blog/cuda-graphs/, (Last accessed, June 27, 2023).

[24] Paras Jain, Xiangxi Mo, Ajay Jain, Harikaran Subbaraj, Rehan Sohail Durrani, Alexey Tumanov, Joseph Gonzalez, and Ion Stoica,Dynamic space-time scheduling for gpu inference.2018

[25] CUDA SAMPLES, https://github.com/NVIDIA/cudasamples/ (Last accessed, June 27, 2023)

[26] 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.

[27] Louis-No¨el Pouchet. 2012. Polybench: The polyhedral benchmark suite. http://www. cs. ucla. edu/pouchet/software/polybench. (Last accessed, June 27, 2023)

[28] Karki, Aajna Keshava, Chethan Shivakumar, Spoorthi Skow, Joshua Hegde, Goutam Jeon, Hyeran. (2019). Tango: A Deep Neural Network Benchmark Suite for Various Accelerators. 137-138. 10.1109ISPASS.2019.00021.

[29] Nsight Compute, https://developer.nvidia.com/nsight-compute (Last accessed, June 27, 2023).

[30] NVIDIA A100 Tensor Core GPU Architecture, https://www.megware.com/fileadmin/user upload/LandingPage%20-NVIDIA/nvidia-ampere-architecture-whitepaper.pdf. (Last accessed, June 27, 2023).

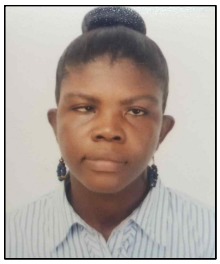
[31]https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html, (Last accessed, June 27, 2023)

[32] NVIDIA MIG User Guide, ttps://docs.nvidia.com/datacenter/tesla/pdf-

/NVIDIA MIG User Guide.pdf, (Last accessed, June 27, 2023).

[33] SCAN, ttps://www.eecs.umich.edu/courses/eecs570/hw/parprefix.pdf
[34] 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)

테오도라 아두푸 (Theodora Adufu)



2013년 : 가나 대학교 컴퓨터과학 및 경제학과 졸업(학사)
2016년: 숙명여자대학교 컴퓨터과학과 졸업(석사)
2022년 ~ 현재: 숙명여자대학교 컴퓨터과학과 (박사과정)

<관심분야> 컨테이너, 클라우드 컴퓨팅, HPC 클라우드, Accelerated Computing(GPUs), DL

하 지 원 (Jiwon Ha)

2023년 8월 : 고려대학교 컴퓨터학과 졸업(학사)
2023년 8월 ~현재 : 서울대학교 컴퓨터공학부 석사과정
<관심분야> Accelerated Computing(GPUs), 병렬처리, DL

김 윤 희 (Yoonhee Kim)

1991년 : 숙명여자대학교 전산학과 졸업(학사)
1996년 : Syracuse University 전산학과 졸업(석사)
2003년: Syracuse University 전산학과 졸업(박사)
1991년~ 1994년 한국전자통신연구원 연구원.
2000년~2001년 Rochester Institute of Technology 컴퓨터공학과 조교수.
2001년 ~ 2016년 숙명여자대학교 컴퓨터과학부 교수.
2017년 ~ 현재 숙명여자대학교 소프트웨어학부 교수.
<관심분야> 클라우드 컴퓨팅, 워크플로우 제어, HPC 클라우드, Accelerated Computing(GPUs)