Application-aware Resource Sharing using Software and Hardware Partitioning on Modern GPUs

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@snu.ac.kr

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

***Abstract*—Graphic Processing Units (GPUs) are known for the large computing capabilities they offer users compared to traditional CPUs. However, the issue of resource under-utilization is becoming more apparent as more and more applications are unable to saturate modern GPUs which have even higher processing capabilities. While concurrency mechanisms like hardware partitioning have resulted in better utilization compared to deployments without sharing, the issue of resource under-utilization still persists even in deployment scenarios where applications are executed on the smallest GPU partitions of modern GPUs. Software partitioning on the other hand, does not guarantee isolation during executions leading to issues of interference and consequently limiting the number of applications which can be run concurrently. Leveraging both software and hardware resource partitioning schemes in an effort to mitigate resource under-utilization issues is yet to be fully explored. In this paper, we evaluate the predictions of a proposed linear regression model relative to actual executions. The results of our experiments show that whilst our approach accurately estimates performance for sharing differently-sized GPU partitions among diverse applications based on each application's characteristics, it also improves utilization and reduces resource wastage.**

***Index Terms*—Resource sharing, resource under-utilization, concurrency, hardware partitioning**

I. Introduction

Computing resources like Graphic Processing Units (GPUs) are gradually becoming part of the mainstream infrastructure in cloud environments [1]–[8], in response to the demand for higher processing power for the execution of Artificial Intelligence (AI) and High Performance Computing (HPC) applications amongst others. Modern GPUs like NVIDIA A100 [9] and H100 [10] GPUs promise to offer users higher computational capacities through the enhanced compute and memory resources they provide. However, they may not always be cost-effective in the absence of options for concurrent executions.

While there are applications which require lots of GPUs [11]–[13] in order to complete execution in a reasonable amount of time, most general purpose GPU applications are unable to saturate the GPU resources dedicated to them [14]–[22]. This results in resource under-utilization at higher overall infrastructural costs to the users who request for GPU cloud resources.

Concurrency mechanisms like CUDA streams [23] [24], Hyper-Q, Multi-Process Service (MPS) [25] as well as Multi-Instance GPU (MIG) [26] have been introduced to implement concurrent executions on GPU resources and hence improve utilization. However these are implemented in isolation and do not consider the characteristics of the applications being executed.

This paper seeks to investigate application-aware approaches to allocating resources to HPC applications using a hybrid of concurrency mechanisms. We

- investigate resource utilization using an application-aware resource allocation approach that leverages sharing at both software and hardware levels
- provide a preliminary heuristic approach to selecting the set of applications to co-run on different GPU partitions.
- propose a resource allocation policy using linear regression modeling in a manner that maximizes resource utilization, reduces interference and reduces wastage of GPU resources

The rest of the paper is organized as follows: in Section 2, we briefly discuss some related works and explain our motivations in Section 3. We then formulate a performance model in Section 4. In Section 5, we evaluate our model with experiments using HPC applications and conclude the paper in Section 6.

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

II. RELATED WORKS

Prior research works like [28]–[32] implement intra-SM resource sharing approaches to improve SM utilization. Chen et al. [33] leverage the compiler to construct GPU tasks and allocate GPU resources uniformly at each kernel launch per request. However, these require modifying the underlying CUDA code of the application and may not be suitable for cloud environments.

Dhakal et al. [17] and Choi et al. [34], determined the share of GPU resources to allocate to each of the two inference models they were co-locating on each GPU using the idea of diminishing marginal returns. Li et al. [35], first share GPU resources using MPS to determine the required SM resources per application and estimate the optimal MIG partition required for the execution of a mix of jobs. Arima et al [36] propose a performance model to optimize resource partitioning, job allocations and power efficiency for HPC applications on MIG-enabled GPU devices. They considered the application's run time characteristics and the influence of power caps in their selection of co-running applications in a manner that improves performance. However, they focus on power allocations and do not consider software partitioning.

III. MOTIVATION

A. GPU Resource Under-utilization

Prior research works observed resource under-utilization when executing Deep Learning (DL) applications [14]–[17] on full GPUs using recent concurrency mechanisms like MPS and MIG. In our previous research on GPU sharing among HPC applications [37], [38], we confirmed through case study based experiments with selected HPC applications from CUDA samples [39], Rodinia [40] and Polybench [41] benchmarks, the issue of resource under-utilization.

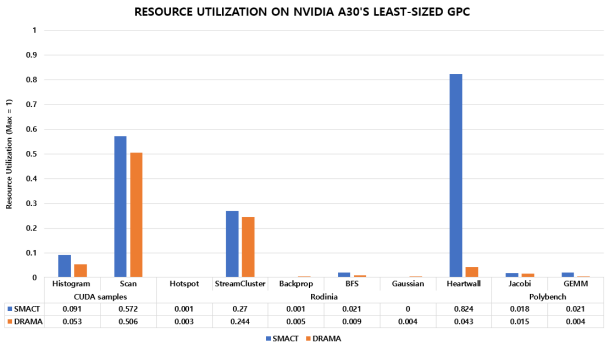

Fig. 1. Resource under-utilization of selected HPC applications

We present in Figure 1, the peak **SM Act**ivity(SMACT) and **DRAM A**ctivity(DRAMA) of some selected applications when executed in solo on the least sized MIG partition(1 GPC) of the NVIDIA A30 GPU. We observed that, with the default execution, partitions(chunks) of GPU resources are not allocated based on each application's resource needs thus there was resource under-utilization even in the smallest hardware partition(1 GPC) of the NVIDIA A30 GPU. Moreover, different applications utilize GPU resources differently and thus the amount of resources which become waste or fragmented [42] varies across applications.

B. Analysis of HPC Application Sensitivity to Resource Allocations

We investigated the performance of different HPC applications when various allocations of GPU resources are allocated to them at the software and hardware levels. We executed each application in solo for different allocations of GPU resources at both the software (MPS) and hardware level (MIG).

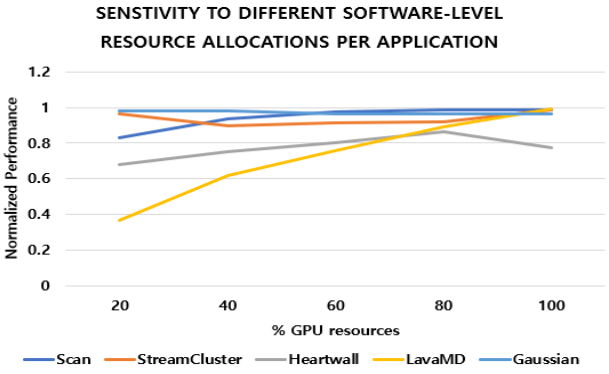

Fig. 2. Resource sensitivity for different software level GPU allocations

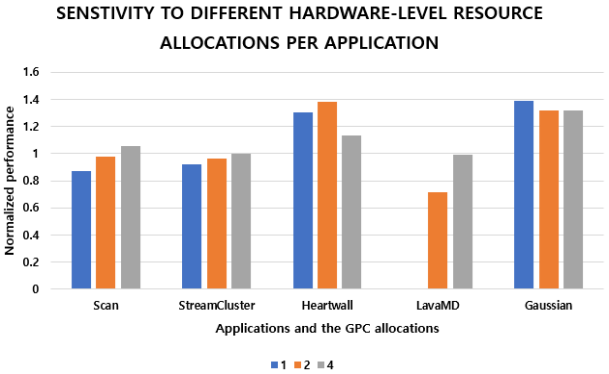

Fig. 3. Resource sensitivity for different hardware level GPU partitions

We observed in Figure 2 that with software partitions, apart from LavaMD [40] which is a heavy-compute intensive(C_h) application, the performance of other applications did not change significantly with the

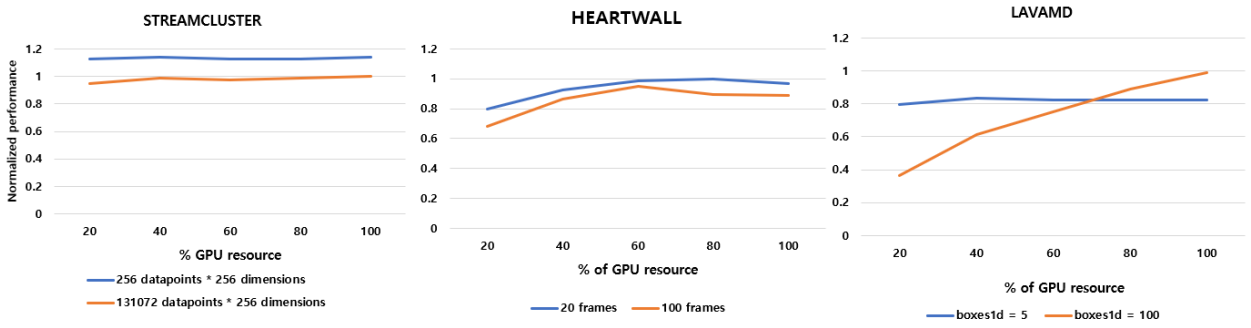

Fig. 4. Application's responsiveness to different resource allocations for different inputs

allocation of more resources. Light-compute intensive (C_l) applications like Heartwall [40] for instance had a "sweet-spot"(40%) beyond which additional resources did not significantly improve performance presenting opportunities for GPU sharing. Unscalable (US) applications like Gaussian [40] also did not respond to increased resources and so could share resources with other applications. With hardware partitions (Figure 3), applications perform differently on different allocations of GPU partitions. For instance, LavaMD did not run on 1 GPC allocation(least-sized GPU partition on NVIDIA A30) since it required more memory and cache resources than the resources provisioned in the partition. We observed however that this was dependent on the input arguments used during the execution. We thus investigated the impact of different input sizes for some applications.

We observed in Figure 4 that, Memory intensive(M) applications like StreamCluster and Light-compute intensive (C_l) applications like Heartwall had similar responses to resource allocation for different input sizes indicating that the input size had negligible impact on performance. Heavy-compute intensive (C_h)applications like LavaMD however were seen to respond more sensitively to resource allocations indicating that for larger inputs, resources should be fully dedicated to such applications. With the observed changes in the sensitivity to resource allocations for different input sizes, we considered input sizes, resource sensitivity, resource utilization, application's profiled characteristics and resource allocations as factors which affect performance.

IV. Opportunities for Co-Sharing GPU Partitions

In a previous study [37], [38], we observed that, by co-scheduling applications with different characteristics in different GPU hardware partitions, more applications could be scheduled and the GPUs overall resource utilization improved. However, deciding which applications to co-run on the GPU partitions requires a consideration of each application's characteristics.

A. Problem Formulation

We extend an optimization problem [36] that enables us to estimate the performance of applications to be co-run using software level sharing on various hardware partitions(*chunk*). Given a **Batch** of applications, the hardware partitions(*chunk*) of the GPU and a fairness parameter (α), we maximize the throughput of the system with regards to the set of applications co-run on the *chunk*, subject to a fairness constraint as follows:

given **Batch**(App_1, App_2, ..., App_n), *chunk*, α
max **T**(App_{set_1}, App_{set_2}, ..., *chunk*, App_x)
s.t. **Fairness**(App_{set_1}, App_{set_2}, ..., *chunk*) $> \alpha$
Output App_{set_i}, $chunk_i$

Our proposed model takes into account the different possible configurations or hardware partitions ($chunk_n$) available on different GPU architectures and each application's characteristics(App_x) obtained through profiling and converted into a vector using basis functions. It then outputs the best $chunk_i$ and App_{set_i} which maximize throughput as well as optimize resource utilization. In this experiment, each App_{set_i} contains only two (2) applications though with current software level concurrency mechanisms, more applications can be co-run per $chunk_i$.

We determine the applications in each App_{set_i} heuristically [28] based on the application's characteristics. We evaluate the Turnaround Time(TT) for a possible pair of applications(TT_{App_i} and $TT_{App_{i+1}}$) in the **Batch**, to determine which applications can be co-run and which ones should be run in solo per App_{set_i}.

The Turnaround Time [28], for consecutive solo-runs on the least GPU partition (1 GPC) is

$$TT_{solo} = (TT_{App_i} + TT_{App_{i+1}})$$

and that for co-execution scenarios is

$$TT_{co} = \max(TT'_{App_i}, TT'_{App_{i+1}}).$$

Using the Average Normalized Turnaround Time (ANTT) [28], [43], we determine that ANTT(TT_{co}/TT_{solo}) less than one (1) indicates better throughput from co-sharing App_i and App_{i+1} thus both applications can be evaluated for performance as an App_{set} using the proposed performance model. By so-doing, we account for interference between the applications to be co-shared.

We maximize the *Throughput, T* of co-running the applications on each $chunk_i$ under a fairness constraint *Fairness* $> \alpha$. *Throughput, T* here, is obtained using the weighted speed-up (*WS*) [36], [44] of all the pairs of applications (App_{set_i}) run on various chunks of the GPU ($chunk_i$) as follows:

$$\begin{aligned}&\mathbf{T}(App_{set1}, App_{set2}, ..., \textit{chunk}, App_x)\\&= \textit{WS}(App_{set1}, App_{set2}, ..., \textit{chunk}, App_x)\\&= \sum_{i=0}^{i<n} RPerf_{App_{seti}}(chunk_i, App_x)\end{aligned}$$

$RPerf_{App_{seti}}(chunk_i, App_x)$ is the relative performance of App_{seti} when executed on $chunk_i$ normalized to performance of App_{seti} when run on full GPU resources.

The fairness constraint, *Fairness* $> \alpha$, is to ensure that App_{seti} with relatively similar performance are co-run. We define *Fairness* as:

$$\begin{aligned}&\textit{Fairness}(App_{set1}, App_{set2}, ..., \textit{chunk}, App_x)\\&= min[RPerf_{App_{set_i}}(), ..., RPerf_{App_{set_n}}()]\end{aligned}$$

B. PERFORMANCE MODEL

We estimate the performance of the applications in the *Batch* with consideration to the computing resources $chunk_i$ allocated to each App_{seti} and the characteristics (App_{xseti}) of the applications. This makes up the first term $A(chunk_i) \cdot B(App_{xseti})$ of our model. The first term also includes a consideration of the possible interference caused by co-running applications within the same $chunk_i$ when applications are selected heuristically for each App_{seti} based on their ANTT.

Additionally, though hardware partitions guarantee resource isolation between *chunks*, modern GPUs have the option of sharing memory resources. In such a case, configured $chunk_i$ are prone to interference. For such a configuration [36], we generalize the relative interference with the second term. We do this with reference to the characteristics of the applications in the App_{setn}.

We model the relative performance of a set of applications executed on the $chunk$ using linear regression performance modeling [36], [45], [46] as follows:

$$\begin{aligned}&RPerf_{App_{seti}}(chunk_i, App_x)\\&= A(chunk_i)\cdot B(App_{xseti}) + \sum_{i\neq n} C(chunk_n)\cdot D(App_{xsetn})\end{aligned}$$

As in [36], the coefficient vectors (A, C) are obtained independently using the Least Square Method (LSM). The vectors $B(App_{xseti})$ and $D(App_{xsetn})$ on the other hand are obtained through conversions with pre-defined basis functions.

V. EXPERIMENT AND EVALUATION

A. Experimental Set-up

We conducted our investigations using selected HPC applications taken from CUDA samples [39], Rodinia [40], Polybench [41] and Tango [50] benchmarks on the NVIDIA A30 GPU.

The applications in App_{setn} are co-run on the $chunk_n$ based on the possible MIG profile configurations on the NVIDIA A30 GPU [26]. This can be modified for the various architectures which have GPU hardware partitioning available (e.g. NVIDIA A100, H100).

TABLE I
MODIFIED PERFORMANCE COUNTERS AND FUNCTIONS [36]

Performance Counters and Resource Utilization Definitions
App_{x1}= SMACT App_{x2}= DRAMA App_{x3}= SMOCC App_{x4}= Compute Throughput [%], App_{x5}= Memory Throughput [%], App_{x6}= L2 Cache size[MB], App_{x7}= Device to L2 Data Writes[MB], App_{x8}= Tensor (mixed) [%], App_{x9}= Tensor (double)[%], App_{x10}= Tensor (integer) [%] App_{x11}= L2 Hit Rate [%], App_{x12}= Turnaround time(solo) [s], App_{x13}= Turnaround time(corun) [s]
Basis Functions $B(App_{xseti})$**,** $D(App_{xsetn})$
B1(memory/compute ratio) = App_{x5}/App_{x4}, B2(Resource sensitivity) = (δ utilization / δ Resource allocation), B3(Cache Utilization) = App_{x7}/App_{x6}, B4(ANTT) = App_{x13}/App_{x12} B5 = const, B6(tensor compute intensity) = (App_{x8} + App_{x9} + App_{x10})/100, B7(non tensor compute intensity) = App_{x4}/100 - B6
D1 = App_{x11}/100 (access pattern related), D2 = const.

To evaluate the co-scheduling scenarios in the hardware partitions, we first collected performance counter values for the applications during a solo run on the full GPU resource using Nsight Compute [48] with some modifications to the setup in [36]. Additionally, we collected metrics on the **SM act**ivity (SMACT), the memory bandwidth utilization or **DRAM a**ctivity (DRAMA) and **SM occ**upancy (SMOCC) every 100ms using NVIDIA's Data Center GPU Manager (DCGM) [47] for both co-run and solo run instances as shown in Table I.

We then classified the applications [30], [36] as shown in Table II, into Heavy Memory intensive(M_h), Light Memory intensive(M_l), Heavy Compute intensive(C_h), Light Compute intensive(C_l), Unscalable(US), using profiled information. Applications which had App_{x3} = 0, or had App_{x1} and App_{x2} = 0 when executed on the smallest hardware partition (1 GPC) were considered Unscalable (US) as they did not utilize a significant amount of the GPU (SM and DRAM) resources. An

TABLE II
APPLICATION CHARACTERIZATION

Application	App_x1	App_x2	App_x3	App_x4	App_x5	**B1**	**B3**	**Class**
Histogram	0.091	0.053	0.086	45.65	94.62	2.07	2.80	M_l
Scan	0.572	0.506	0.536	13.05	79.29	6.08	1.14	M_h
Backprop	0.001	0.005	0.001	60.92	61.32	1.01	0.38	C_l
BFS	0.021	0.009	0.015	16.43	10.1	0.61	0.04	C_l
Gaussian	0	0.004	0	0.04	0.59	14.75	0	US
Heartwall	0.824	0.043	0.299	7.82	6.19	0.79	0.01	C_l
Hotspot	0.001	0.003	0	71.51	30.85	0.43	0.09	US
StreamCluster	0.27	0.244	0.242	22.76	77.13	3.39	3.01	M_l
GEMM	0.018	0.015	0.021	23.81	39.24	1.65	0.13	C_l
Jacobi	0.021	0.004	0.015	30.88	37.83	1.26	0.17	M_l
LavaMD(2GPC)	1	1	0.562	94.95	16.12	0.17	82.92	C_h

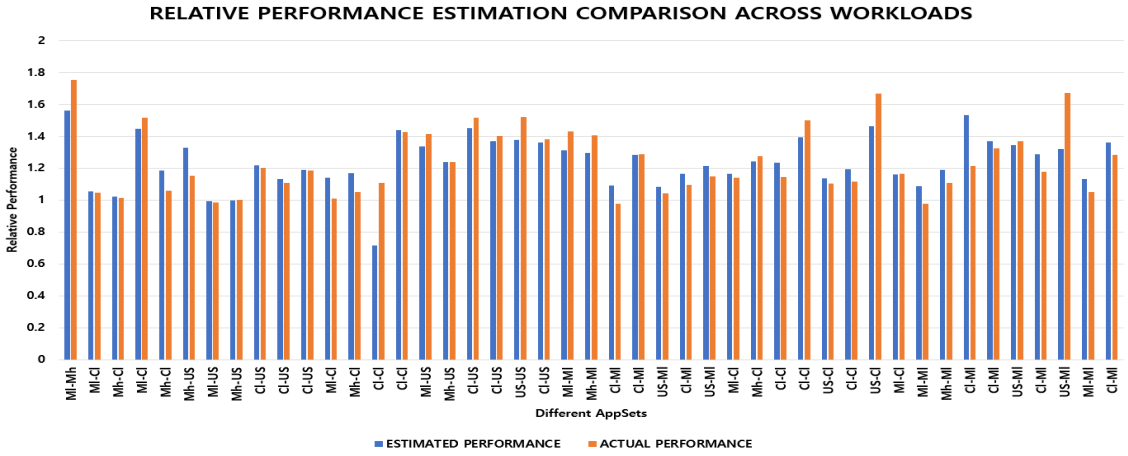

Fig. 5. Relative Performance Estimation Comparison Across Different Workloads

application was classified as memory intensive if the memory throughput to compute throughput ratio (B1) was more than one(1.0), and compute intensive otherwise. Heavy Compute intensive applications were those which scaled well with more SM resources and had high SM resource utilization whilst Light Compute intensive had lower resource utilization and were less responsive to allocations. Similarly, Heavy memory intensive applications were those which had significant bandwidth utilization whilst Light memory intensive had lower bandwidth utilization. Applications whose cache utilization was >1 were also expected to be allocated sufficient resources. In this evaluation, we did not consider TensorCore intensiveness due to the selected benchmarks used for the study however this could be included by evaluating related performance counter statics App_{x8}, App_{x9}, App_{x10} obtained during profiling.

After characterizing the applications, we heuristically [28] selected the sets of applications in the $Batch$, estimated the relative performance and executed them on the available chunks in our experiments. We however gave higher priority to applications characterized as Heavy Memory Intensity(C_h) or Heavy Compute Intensity(C_h) and the least priority to Unscalable(US) applications.

B. Experiment Evaluation

We begun by evaluating the model's estimation of relative performance ($RPerf_{App_{seti}}(chunk_i, App_x)$) for App_{seti} through co-running a combination of the first 10 applications $\binom{10}{2}$ (Table II) on a $chunk(1GPC)$ as shown in Figure 5. For clarity in the graph, we represented each App_{seti} with their characteristics.

Our proposed model estimated the relative performance of applications on GPU $chunks$ well with respect to the application characteristics of all the application's within the App_{seti} thus we evaluated the model for it's estimation of throughput for different configuration of $chunks$ and for different workloads in isolated $chunks$.

TABLE III
RESOURCE UTILIZATION COMPARISON ACROSS DIFFERENT WORKLOADS FOR DEFAULT MIG AND THE PROPOSED APPROACH

Batch	Hardware partition	MIG Default Execution (1:1)				Proposed approach (MIG + MPS)			
		Applications	SMACT	SMOCC	DRAMA	Applications	SMACT	SMOCC	DRAMA
B_1	1 -1g.6gb	Scan	0.578	0.538	0.509	SCAN + StreamCluster	0.625	0.577	0.551
	2 -1g.6gb	StreamCluster	0.273	0.244	0.247	GEMM	0	0	0.001
	3 -1g.6gb	Gaussian	0	0	0.004	Gaussian	0	0	0.004
	4 -1g.6gb	GEMM	0	0	0	**FREE RESOURCE**			
B_2	1 -1g.6gb	Heartwall	0.761	0.281	0.037	Heartwall + Hotspot	0.862	0.316	0.043
	2 -1g.6gb	BFS	0.022	0.017	0.01	BFS	0.023	0.017	0.008
	3 -1g.6gb	Histogram	0.089	0.085	0.052	Histogram	0.09	0.085	0.052
	4 -1g.6gb	Hotspot	0.001	0	0.003	**FREE RESOURCE**			
B_3	1 -2g.12gb(1st run)	LavaMD	1	0.562	0.435	LavaMD	1	0.562	0.086
	2 -2g.12gb(1st run)	Gaussian	0.005	0.004	0.003	Scan + Jacobi	0.517	0.472	0.451
	1 -2g.12gb(2nd run)	Jacobi	0	0	0	Gaussian	0	0	0.003
	2 -2g.12gb(2nd run)	Scan	0.511	0.467	0.446	**FREE RESOURCE**			

We evaluate the proposed approach for three (3) $Batch$ of applications B_1(Scan, StreamCluster, Gaussian, GEMM), B_2(Heartwall, BFS, Histogram, Hotspot) B_3(Scan, LavaMD, Gaussian, Jacobi) submitted to a GPU with homogeneous $chunks$ of MIG 1g.6gb and 2 MIG 2g.12gb respectively. We also compared the proposed approach to the default MIG execution(1 App_x: 1 $chunk$). With the proposed approach, the maximum possible window of applications in each submitted $Batch$ is (2 * num_$chunk$), however, we choose a window of four (4) diverse applications per $Batch$ for the evaluation.

OBSERVATION 1: IMPROVED THROUGHPUT WITH CO-SHARING

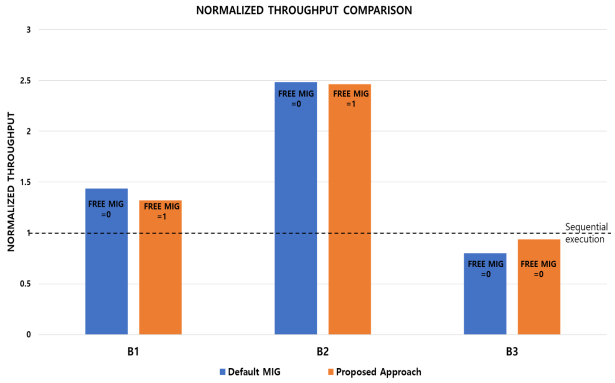

Fig. 6. Throughput Comparison Across Different Workloads

From Figure 6, the normalized throughput of the default MIG and the proposed approach showed performance improvements of approximately 43.7% and 32% for B_1 and 148.6% and 146.3% for B_2, respectively, compared to sequential execution. For B_3, both the default MIG and the proposed approach showed a performance reduction of 20% and 6.22% respectively. The relatively lower throughput obtained for B_3 was due to the fact that our proposed approach suggested dedicating a $chunk$ to $heavy$ applications. Consequently we dedicated a $chunk$(2 GPC) to the C_h application(LavaMD) and run the third application immediately after the execution of the first App_{set}. Compared to the default MIG partitioning however, the proposed approach performed better by about 13.78%.

OBSERVATION 2: IMPROVED RESOURCE UTILIZATION WITH CO-SHARING

We also observed (Table III, Figure 6) that, our proposed approach reduced the number of $chunks$ used during the execution per $Batch$ compared to default MIG (1 App_x**:**1 $chunk$) approach thus freeing resources for use by other applications. From Table III, SMACT, SMOCC and DRAMA utilization improve when co-running applications in $chunk$ as we were able to pre-determine which set of applications when executed together would improve turnaround time on fewer resources. This suggested that, in scenarios where there are more applications than available resources, our proposed approach would be able to improve the number of executions with fewer resources.

VI. CONCLUSION AND FUTURE WORKS

This paper investigated application-aware resource sharing using a hybrid of software and hardware GPU sharing mechanisms. Our proposed model takes into account characteristics of the applications and the possible hardware partitions ($chunk$) per GPU to determine the App_{set} to co-run using software partitioning. From evaluations, our model is able to accurately estimate relative performance of an App_{set}, reduce the number of resources used for each $Batch$ and improve utilization. Next, we intend to investigate dynamic resource scheduling based on decisions made using the proposed approach.

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] IBM Cloud Server's NVIDIA GPU, https://www.ibm.com/kr-ko/cloud/gpu?mhsrc=ibmsearch_a&mhq=GPU, (Last accessed, October 17, 2023)

[2] Recommended GPU instances, https://docs.aws.amazon.com/ko_kr/dl-ami/latest/devguide/gpu.html, (Last accessed, October 17, 2023)

[3] Cloud GPU, https://cloud.google.com/gpu?hl=ko, (Last accessed, October 17, 2023)

[4] HPC in the cloud, https://www.ni-sp.com/support/hpc-in-the-cloud/ (Last accessed, October 17, 2023)

[5] Elastic GPU, https://www.alibabacloud.com/ko/product/gpu, (Last accessed, October 17, 2023)

[6] Cloud computing, https://aws.amazon.com/what-is-cloud-computing/ (Last accessed, October 17, 2023)

[7] https://www.run.ai/guides/multi-gpu/gpu-scheduling

[8] GPU Cloud Computing, https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/ (Last accessed, October 17, 2023)

[9] NVIDIA DGX A100 Datasheet, https://resources.nvidia.com/en-us-dgx-systems/dgx-ai (Last accessed, October 17, 2023)

[10] NVIDIA H100 Tensor Core GPU Datasheet, https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet (Last accessed, October 17, 2023)

[11] Carminati Federico, Vallecorsa Sofia, Khattak Gulrukh, Codreanu Valeriu, Podareanu Damian, Cai Maxwell, Saletore Vikram, Pabst Hans. (2020). Generative Adversarial Networks for Fast Simulation: distributed training and generalisation. 012. 10.22323/1.372.0012.

[12] Efficient Access to Shared GPU Resources: Part 1, https://kubernetes.web.cern.ch/blog/2023/01/09/efficient-access-to-shared-gpu-resources-part-1/ (Last accessed, October 17, 2023)

[13] Duarte Joel, Vlimant Jean-Roch. (2022). Graph Neural Networks for Particle Tracking and Reconstruction. 10.1142/9789811234026_0012.

[14] 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.

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

[16] 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.

[17] Aditya Dhakal, Sameer G Kulkarni, K. K. Ramakrishnan. "GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform." [Online]. Available: [URL]

[18] 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

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

[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] 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

[23] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (Last accessed, October 17, 2023)

[24] https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/ (Last accessed, October 17, 2023)

[25] NVIDIA, https://docs.nvidia.com/deploy/mps/index.html(2020), (Last accessed, October 17, 2023)

[26] NVIDIA Multi-Instance GPUs, https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html (Last accessed, October 17, 2023).

[27] NVIDIA MIG User Guide, https://docs.nvidia.com/datacenter/tesla/pdf-/NVIDIA_MIG_User_Guide.pdf, (Last accessed, October 17, 2023).

[28] Allen, T., Feng, X., Ge, R.: Slate: enabling workload-aware efficient multiprocessing for modern gpgpus. In: 2019 IEEE international parallel and distributed processing symposium (IPDPS), pp. 252–261. IEEE

[29] Qichen Chen, Hyerin Chung, Yongseok Son, Yoonhee Kim, Heon Young Yeom, "smCompactor: AWorkload-aware Fine-grained Resource Management Framework for GPGPUs", SAC'21, March 22-March 26, 2021,

[30] Sejin Kim, Yoonhee Kim,"K-Scheduler: dynamic intra-SM multitasking management with execution profiles on GPUs", https://link.springer.com/article/10.1007/s10586-021-03429-7

[31] Dai, H., Lin, Z., Li, C., Zhao, C., Wang, F., Zheng, N., Zhou, H.: Accelerate gpu concurrent kernel execution by mitigating memory pipeline stalls. In: 2018 IEEE international symposium on high performance computer architecture (HPCA), pp. 208–220. IEEE (2018)

[32] Xu, Q., Jeon, H., Kim, K., Ro, W.W., Annavaram, M. Warped-slicer.: Efficient intra-sm slicing through dynamic resource partitioning for gpu multiprogramming. In: 2016 ACM/IEEE 43rd annual international symposium on computer architecture (ISCA) (2016), pp. 230–242. IEEE (2016)

[33] Chao Chen, Chris Porter, and Santosh Pande. 2022. CASE: a compiler-assisted SchEduling framework for multi-GPU systems. In Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '22). Association for Computing Machinery, New York, NY, USA, 17–31. https://doi.org/10.1145/3503221.3508423

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

[35] 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

[36] Eishi Arima, Minjoon Kang, Issa Saba, Josef Weidendorfer, Carsten Trinitis, and Martin Schulz. 2023. Optimizing Hardware Resource Partitioning and Job Allocations on Modern GPUs under Power Caps. In Workshop Proceedings of the 51st International Conference on Parallel Processing (ICPP Workshops '22). Association for Computing Machinery, New York, NY, USA, Article 9, 1–10. https://doi.org/10.1145/3547276.3548630

[37] Theodora Adufu, Jiwon Ha, Yoonhee Kim, "An Analysis of Efficient GPU Resource Sharing for Concurrent HPC Application Executions", KNOMS Review, Vol.25, No. 1, September, 2022

[38] Theodora Adufu, Jiwon Ha, Yoonhee Kim, "Exploring the Diversity of Multiple Job Deployments over GPUs for Efficient Resource Sharing",

[39] CUDA SAMPLES, https://github.com/NVIDIA/cudasamples/ (Last accessed, October 17, 2023)

[40] 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.

[41] Louis-Noël Pouchet. 2012. Polybench: The polyhedral benchmark suite. http://www. cs. ucla. edu/pouchet/software/polybench. (Last accessed, October 17, 2023)

[42] 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

[43] Hiroshi Sasaki, Teruo Tanimoto, Koji Inoue, and Hiroshi Nakamura. 2012. "Scalability-Based Manycore Partitioning". In Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT). 107–116.

[44] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das. 2009. Application-Aware Prioritization Mechanisms for On-Chip Networks. In 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 280–291.

[45] Eishi Arima, Toshihiro Hanawa, Carsten Trinitis, and Martin Schulz. 2020. Footprint-Aware Power Capping for Hybrid Memory Based Systems. In International Conference on High Performance Computing. 347–369.

[46] Hiroshi Sasaki, Yoshimichi Ikeda, Masaaki Kondo, and Hiroshi Nakamura. 2007. An Intra-Task Dvfs Technique Based on Statistical Analysis of Hardware Events. In 4th International Conference on Computing Frontiers (CF). 123–130.

[47] NVIDIA, Data Center GPU Manager (DCGM) 3.1, https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/feature-overview.html (Last accessed, October 17, 2023)

[48] Nsight Compute, https://developer.nvidia.com/nsight-compute (Last accessed, October 17, 2023).

[49] J. Alsop et al., “Optimizing GPU Cache Policies for MI Workloads” 2019 IEEE International Symposium on Workload Characterization (IISWC), 2019, pp. 243-248, doi:10.1109/IISWC47752.2019.9041977.

[50] 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.