Serverless Graph Analytics on Multi-Instance GPU

Mohammad Sonji
Department of Computer Science
American University of Beirut
Beirut, Lebanon
mms158@mail.aub.edu

Mohammed Baydoun
Department of Computer Science
American University of Beirut
Beirut, Lebanon
mb57@aub.edu.lb

Aditya Dhakal
Hewlett Packard Enterprise
Milpitas, CA, USA
aditya.dhakal@hpe.com

Gourav Rattihalli
Hewlett Packard Enterprise
Milpitas, CA, USA
gourav.rattihalli@hpe.com

Dejan Milojicic
Hewlett Packard Enterprise
Milpitas, CA, USA
dejan.milojicic@hpe.com

Izzat El Hajj
Department of Computer Science
American University of Beirut
Beirut, Lebanon
izzat.elhajj@aub.edu.lb

***Abstract*—Serverless computing is a promising paradigm because it provides scalability and cost-efficiency to cloud users, while reducing the burden of infrastructure management. Most mature serverless computing solutions target CPUs, with fewer solutions targeting GPUs and typically being focused on machine learning workloads. In this paper, we explore the effectiveness of serverless computing for graph analytics workloads running on GPUs, particularly multi-instance GPU. We leverage the Knative serverless computing framework and integrate Tigr as the main GPU graph analytics engine. We optimize the placement of the graph datasets used by the serverless functions, keeping many warm in the CPU main memory and a few warm in the GPU memory based on their frequency of use. We evaluate the latency and throughput of the serverless functions, showing how the best choice of MIG configuration is impacted by the choice of function, dataset, and workload heterogeneity.**

I. Introduction

Serverless computing has become a popular computing paradigm because it provides scalability and cost-efficiency via its fine-grained pay-per-use billing model, while also alleviating the need for users to manage server infrastructure [1]. It also allows cloud providers to share resources among multiple users more efficiently. Numerous mature serverless computing solutions are available from major cloud providers [2]–[4], but these solutions are mostly focused on CPU systems. Serverless computing using GPUs is still a maturing technology, and multi-instance GPU [5] (MIG), whereby a GPU is partitioned into multiple isolated instances that can be shared by different users, is a natural feature to consider when designing a serverless solution for GPUs.

There have been multiple prior works proposing serverless computing frameworks for GPUs [6]–[17], with a number of them targeting multi-instance GPU [18]–[20]. Many of these prior works cater to machine learning workloads due to the prevalence of GPUs in accelerating machine learning. Graph analytics is also an important computation on GPUs [21]–[23], however, none of the prior serverless frameworks for graph analytics target GPUs [24]–[28]. To the best of our knowledge, there is no prior work that studies the interaction between serverless computing and graph analytics on GPUs.

To fill this gap, we present an evaluation of the serverless computing paradigm applied to graph analytics workloads running on multi-instance GPUs. Our setup leverages Knative [29], a serverless computing framework based on Kubernetes. For graph analytics tasks on GPUs, we use Tigr [21], a performant and lightweight graph transformation and processing framework for GPUs, and extend it to handle graph processing requests as a set of serverless functions. To support GPUs, we use the NVIDIA GPU Operator [30] which automates the management of GPU resources within Kubernetes, and is also responsible for setting up and managing MIG configurations on the nodes within a Kubernetes cluster. An important optimization that we apply is keeping frequently used graph datasets cached in the CPU main memory and very frequently used graph datasets cached in the GPU memory.

Our evaluation shows the trade-off between throughput and latency for different MIG configurations, and how the choice of the best MIG configuration is impacted by the choice of function, dataset, and workload heterogeneity. We also evaluate the impact of our optimizations.

Our future work involves evaluating the impact of leveraging such a serverless framework in a broader workflow with many interacting function invocations, since our evaluation focuses primarily on independent invocations. We also propose other future optimizations such as dynamically reconfiguring the MIG instances to respond to changes in workload characteristics, developing performance models for the serverless functions to inform the scheduling and reconfiguration tasks, and evaluating the impact of using heterogeneous MIG configurations.

The rest of this paper is organized as follows. Section II surveys prior work on serverless computing, the use of GPUs, and serverless graph analytics frameworks. Section III describes our setup for performing serverless graph analytics on multi-instance GPUs. Section IV outlines our evaluation methodology and Section V details our evaluation. Finally, Section VI discusses future work and concludes.

II. Related Work

A. Serverless Computing

Serverless computing provides a scalable and cost-efficient way for offloading computations to the cloud without the need to manage server infrastructure [1]. Cloud platforms like AWS Lambda [2], Microsoft Azure Functions [3], Google Cloud Functions [4], and other serverless platforms provide the feature of executing serverless functions based on a certain event, without the need for server management. Kubernetes [31] is an open-source system for automating deployment, scaling, and management of containerized applications [31] that could be used to run on these cloud platforms. Knative [29] and OpenFaaS [32] are some of the Kubernetes frameworks for enabling serverless functions. According to recent surveys about some challenges of serverless computing, the cold start latency problem [33] is an important issue that is being addressed as it can decrease the benefits of serverless computing. Other challenges come from the scheduling algorithms [34] used in such serverless environments which could hurt resource utilization.

B. Serverless Computing with GPUs

Major serverless cloud platforms [2]–[4] have supported mostly CPUs for their serverless functions, with GPUs being supported only recently [35], [36]. Recent research has also explored expanding serverless capabilities to include functions that can leverage the power of GPUs for faster processing [6]–[17]. None of these prior works on serverless GPU computing specifically target graph analytics. Our work is the first to study the interaction between serverless GPU computing and graph analytics.

Some prior works [37], [38] target graph neural network serving using GPUs. Our work focuses on graph analytics, as opposed to graph neural network serving.

C. Serverless Computing with Multi-Instance GPU

An important aspect of serverless frameworks is allowing multiple serverless functions to share the same hardware resources. However, multiple tasks running simultaneously on a single GPU often compete for the same resources, leading to performance interference and possible violation of QoS requirements. NVIDIA GPUs support multiple forms of spacial sharing including CUDA contexts, Multi-Process Server (MPS), and Multi-Instance GPU (MIG) [5]. MIG technology partitions the GPU into multiple instances, each with its own isolated compute and memory resources. This isolated partitioning allows multiple tasks to run independently without interfering with each other in the compute and memory usage, ensuring more consistent performance and QoS. Some prior works [18], [19], [39] leverage GPU MIG partitioning in a serverless environment for machine learning applications. By using MIG, these works offer isolated GPU environments to different users, which addresses the challenge of efficiently sharing expensive GPU resources among multiple tenants. Still, none of these prior works target graph analytics and our work is the first to do so. While MPS can also be used for GPU sharing, it only provides isolation at the level of the cores but not the memory. Since graph analytics workloads tend to be memory-bound, we prefer using MIG to isolate the memory as well, but we leave exploring MPS as future work.

D. Serverless Graph Analytics

A number of prior works offer serverless frameworks for graph analytics [24]–[28]. For example, Graph-Serverlizer [24] is specifically designed to harness the power of serverless computing for efficient and scalable graph processing. It encapsulates basic graph processing operations as serverless functions, optimizing their deployment and execution across a computing continuum. Graphless [25] tackles the incompatibility between the stateless nature of serverless functions and the data storage demands of graph processing. It addresses this issue by introducing a specialized memory storage service (Memory-as-a-Service). None of the prior works on serverless graph analytics target GPU systems specifically, let alone multi-instance GPU. Our work is the first to study serverless graph analytics targeting GPUs.

E. Graph analytics on GPUs

GPUs have been used to accelerate a wide variety of graph algorithms [40]–[45]. There are numerous libraries for performing graph computations on GPUs [21]–[23]. Tigr [21] is the most efficient of these libraries for a limited set of kernels. These prior works on graph analytics on GPUs are used in a standard HPC setup. Our work is the first to study GPU graph analytics in a serverless context. Although we leverage Tigr [21] as the GPU graph analytics engine, our work can easily be extended to other GPU graph analytics libraries as well.

III. Our Setup for Serverless Graph Analytics on Multi-Instance GPUs

Our serverless graph analytics setup for multi-instance GPUs is built upon the Knative [29] serverless computing framework, uses Tigr [21] for graph processing on GPUs, and uses the NVIDIA GPU Operator [30] to enable GPU access and management of MIG instances in Kubernetes. Figure 1 shows an overview of this setup. In this section, we describe our serverless computing setup (Setion III-A), our integration with Tigr (Section III-B), how we support GPUs (Section III-C), and our optimizations for keeping graph datasets cached in warm containers (Section III-D).

A. Serverless Computing Framework

Our serverless graph analytics setup leverages Knative [29] installed on a Kubernetes cluster. We choose Knative because it is open source and had wide community support. We configure the network layer using Kourier [46], a lightweight and efficient ingress controller designed for Knative. We integrated Flannel [47] to ensure reliable networking capabilities within the cluster. To handle the large datasets required for graph processing, we provision a persistent volume. This persistent volume allows pods to access the necessary datasets, ensuring

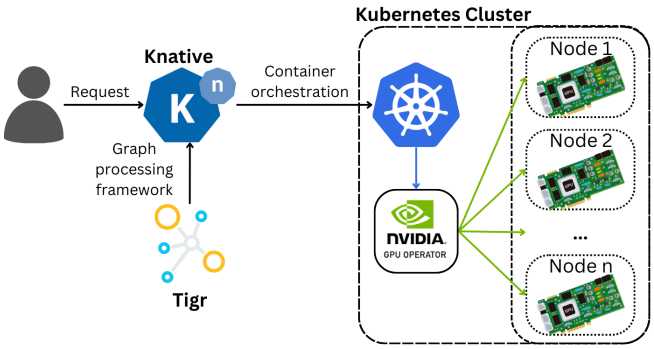

Fig. 1. Overview of our setup for serverless graph analytics on GPUs

that data is readily available when required for processing tasks.

The serverless functions requests we receive must specify the graph analytics operation to be executed and the dataset it should be executed on. The response contains the results of the executed function on the specified dataset. We use JSON as the communication format for its lightweight and human-readable structure, facilitating efficient data exchange between the client and the server.

The serverless functions are represented in a YAML file which specifies multiple attributes that are required by each serverless function such as the scale range of pods for auto-scaling, the container concurrency, the docker image of the serverless graph processing framework (see Section III-B), the GPU MIG resources required, and the persistent volume where the datasets are located.

B. Integration of the Graph Analytics Framework

Tigr is a lightweight graph transformation and processing framework for GPUs that has been shown to be more efficient than other graph processing frameworks for GPUs [21]. We extend Tigr using Crow [48] to handle graph processing requests as a set of serverless functions. Crow manages HTTP requests, routing them to handlers designed to manage the requested graphs and then execute the requested functions. The extended Tigr framework is containerized in a docker image to be used for Knative [29] serverless functions.

With the graph analytics computations happening entirely on the GPU, the role of the CPU in our setup is to listen for and handle the HTTP requests as well as manage the caching of datasets (see Section III-D). However, it may also be beneficial to have the CPU perform some of the computations alongside the GPU to better utilize overall system resources [49], [50]. Such hybrid CPU-GPU execution is the subject of future work.

C. GPU Support in Kubernetes

Kubernetes [31] allows applications to access specialized hardware like GPUs, network adapters, and other devices using device plugins. However, setting up nodes with this hardware requires complex configuration of various software components, which is error-prone. To simplify this process, the NVIDIA GPU Operator [30] automates the management of all necessary NVIDIA software within Kubernetes, which includes installing drivers, configuring GPU access for containers, monitoring hardware performance, and more. We use the NVIDIA GPU Operator to enable GPU access within our Kubernetes cluster.

The GPU Operator is also responsible for setting up and managing MIG configurations on the nodes within a Kubernetes cluster. The MIG manager component of the operator efficiently slices and reconfigures these instances allowing us to reconfigure the geometry of the GPU instances based on workload demand.

D. Keeping Graph Datasets Cached

An important optimization that we apply is keeping the input graph datasets cached in the CPU's main memory as well as the GPU memory as long as the container is warm, so that the datasets do not need to be read from disk and copied to the GPU memory every time. We refer to the graph datasets as being warm if they are cached in memory in a warm container. Keeping the datasets warm is crucial for reducing the data transfer latency from the disk to the CPU main memory and from the CPU main memory to the GPU memory, which could sometimes exceed the latency of the kernel function itself. It ensures that frequently accessed graphs are readily available in memory, reducing the need for repeated data-loading operations.

Figure 2 shows the process for keeping the graph datasets warm. We maintain a list of graph datasets that are warm in the CPU main memory as well as in the GPU memory. When an invocation requests a graph, we first look up if the graph dataset is already in the CPU main memory. If not, we read the graph dataset from storage to the CPU main memory, then copy the graph to the GPU memory and begin the execution. The copies of the graph in the CPU main memory and GPU memory are kept in memory for subsequent invocations as long as the container is warm. Hence, the initial load may introduce some latency, but subsequent requests for the same graph dataset will benefit from reduced load times.

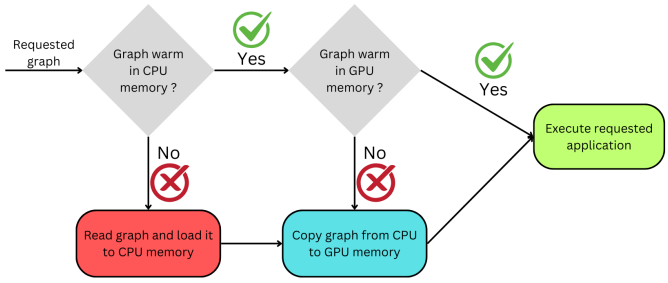

Fig. 2. Process for keeping graph datasets warm

If the graph dataset is already warm in the CPU main memory, we look up if the graph is already warm in the GPU memory. If yes, we simply execute the function. If not, we

transfer the graph from CPU main memory to GPU memory. This transfer is significantly faster than loading the graph from disk.

We keep multiple graph datasets warm at a time in the CPU main memory and the GPU memory, depending on the size of the CPU main memory and the size of the MIG instance's memory. Naturally, MIG instances with larger memory capacity allow for more graphs to be kept warm simultaneously. Keeping multiple graphs warm simultaneously allows for efficient handling of concurrent requests to multiple different graph datasets by reducing the frequency of data transfer. A question that arises when keeping multiple datasets warm is which graph datasets to evict when the memory is full to make way for new graph datasets to be loaded. To decide which graph to evict, we adopt a least recently used replacement policy.

IV. Experimental Methodology

A. Hardware and Software Specifications

The hardware and software specifications of our experimental setup is summarized in Table I.

TABLE I
System Specifications setups.

	Configuration
CPU	AMD EPYC 7543 Processor @ 2.80 GHz, Core(s) per socket: 32, Socket(s): 2
GPU	NVIDIA A100 with 80GB of memory, Driver Version: 550.54.15, CUDA Version: 12.4
Software	Operating system: Ubuntu 22.04.4 LTS, Docker version: 27.0.3
Container	Base Image: nvidia/cuda:12.3.1-base-ubuntu22.04

The MIG instance types considered in this work are shown in Table II. In our experiments, we always partition the GPU into a homogeneous set of MIG instances and use all the instances. For example, when choosing the 1g.10gb instance type, we divide the GPU into seven of these instances and distribute the serverless functions across all seven instances, dispatching each function to whichever instance is available. Hence, there is a trade-of between the number of instances and the resources available to each instance. Selecting the right instance type involves a delicate balance between these two factors, which we evaluate in this work.

TABLE II
MIG Instances and their resources

#Slices x MIG profile	Number (ratio) of SMs	GPU Memory (GB)
7 x 1g.10gb	14 (1/7)	10
4 x 1g.20gb	14 (1/7)	20
3 x 2g.20gb	28 (2/7)	20
2 x 3g.40gb	42 (3/7)	40
1 x 4g.40gb	56 (4/7)	40
1 x 7g.80gb	98 (7/7)	80

It is also possible to partition the GPU into a heterogeneous set of MIG instances. For example, one could configure two 2g.20gb and one 3g.40gb instance. The benefit of heterogeneity is that one could be selective about which instance to send a function to depending on the execution resources the function needs as well as the dataset size. However, such heterogeneous partitioning requires more sophisticated support and is part of our future work.

B. Benchmarks and Datasets

We use the six graph analytics computations available in the Tigr [21] framework:

1) Breadth First Search (BFS)
2) Betweenness Centrality (BC)
3) Connected Components (CC)
4) Pagerank (PR)
5) Single Source Shortest Path (SSSP)
6) Single Source Widest Path (SSWP)

TableIII list the graph datasets used in our evaluation. We use undirected graphs for BC, BFS, CC, and PR, and directed graphs for SSSP and SSWP. Hence, the same graph is considered as two different datasets in the experiments where heterogeneous workloads are used that include a mixture of these functions.

TABLE III
Graph Datasets

Dataset	**Number of Vertices**	**Number of Edges**	**Reference**
com-orkut.ungraph	3,072,441	117,185,083	[51]
soc-sinaweibo	58,655,849	261,321,071	[52]
mawi-1	18,571,154	19,020,160	[53]
mawi-2	35,991,342	37,242,710	[53]
orkut-groupmemberships	8,730,857	327,036,553	[52]
soc-twitter-2010	21,297,772	265,025,809	[52]
cage15	5,154,859	52,177,205	[52]
in-2004	1,382,870	13,968,883	[52]
indochina-2004	7,414,769	153,487,304	[52]
ljournal-2008	5,363,201	50,545,899	[52]
patents	3,750,822	14,970,768	[51]
Stanford_Berkeley	683,446	6,634,677	[52]
wikipedia-20051105	1,598,583	18,557,791	[52]
V2a	55,042,369	58,608,800	[52]
wiki-topcats	1,791,489	25,447,873	[51]
kron_g500-logn21	1,544,087	91,042,010	[52]
eu-2005	862,664	16,639,895	[52]
cage14	1,505,785	14,318,067	[52]

C. Load Generation

For experiments with homogeneous workloads, i.e., where we test each function and graph dataset independently, we used the Hey [54] load generator. Hey is a versatile HTTP load generator primarily used for performance testing of web applications. It allows us to simulate multiple concurrent requests to our serverless framework.

For experiments with heterogeneous workloads, i.e., where the workload consists of multiple different functions and datasets, we employed an asynchronous Python library to simulate invocations from a trace, mimicking real-world usage

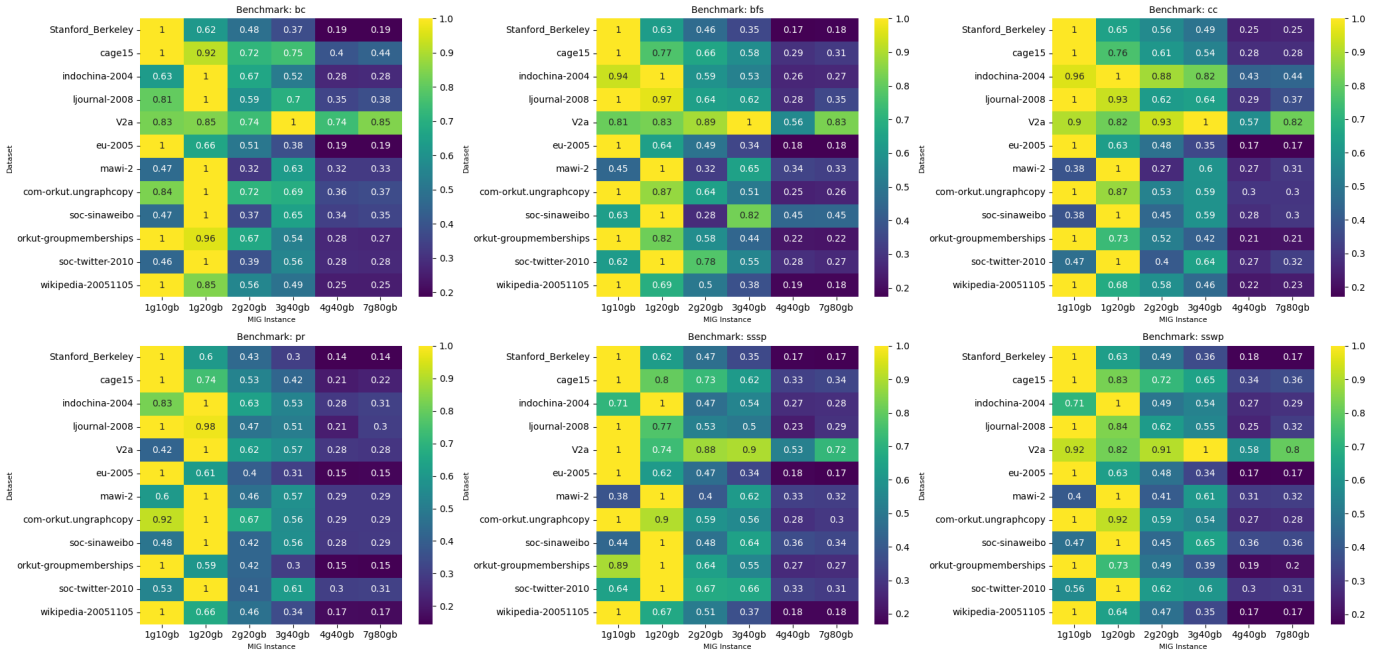

Fig. 3. Normalized throughput (higher is better) for different functions, datasets, and MIG configurations. For each function and dataset, the throughput is normalized to that of the best MIG configuration.

patterns. The trace we used is published by the Systems Infrastructure Research lab at Huawei's Edinburgh Research Center [55]. We used the per-minute trace version from their private cloud. This approach allows us to simulate multiple concurrent invocations, as specified in the trace, ensuring that the timing and order of requests closely followed the trace's pattern. It enables us to evaluate how the framework performs under realistic and complex workload patterns.

V. Evaluation

A. Impact of Function and Dataset on Best MIG Instance

Figure 3 shows how the throughput of processing the serverless functions varies based on the graph analytics operation that is executed, the dataset that is used, and the MIG configuration. The throughput was determined based on 600 executions of the same function with the same dataset. It is calculated as the number of requests processed by all the MIG instances of the given type divided by the makespan of processing all the requests. The heatmaps show the throughput normalized for each function and dataset pair because the absolute throughput depends on the specific choice of function and dataset. The throughput is normalized to that of the best performing MIG instance.

Our key observation from Figure 3 is that the best MIG configuration depends on the function that is being executed, and even more, on the dataset that it executes with. This result motivates the need to model the performance of the functions for different datasets and MIG instance sizes, and to use this information when configuring or reconfiguring the MIG instances, which is the subject of our future work.

To better understand what causes a particular MIG configuration to have the best throughput, Figure 4 shows how the average latency, average execution cost (core-hours), and throughput of the serverless functions varies across MIG configurations for the BFS function executed with the V2a dataset as an example. The latency and throughput are both measured, whereas the execution cost is derived from the average latency and is inversely proportional to efficiency. We calculate execution cost by multiplying the effective number of SM slices reserved by an instance with the latency of the computation executed on that instance. For example, the 1g.10gb instance uses one SM slice and also effectively reserves one SM slice. However, the 1g.20gb instance uses one SM slice, but there are only four such instances reserving all seven SM slices, so the 1g.20gb instance effectively reserves (7 SM slices)/(4 instances) = 1.75 SM slices per instance.

Our key observation from Figure 4 is that as the MIG instances get larger, the latency decreases proportionally or superlinearly for the initial MIG configurations. For this reason, the execution cost decreases initially and the throughput increases because the improvement in latency per instance outweighs the decrease in the number of instances being used concurrently. However, the latency improvement faces diminishing returns for the larger MIG configurations beyond 3g.40gb. As a result, the execution cost increases again and the throughput decreases because the marginal reduction in latency is not sufficient to compensate for having fewer instances. Therefore, the measured throughput is highest at 3g.40gb and appears to be inversely proportional to the execution cost, which is expected because we are testing a fully utilized system at steady state. The throughput is maximum when the

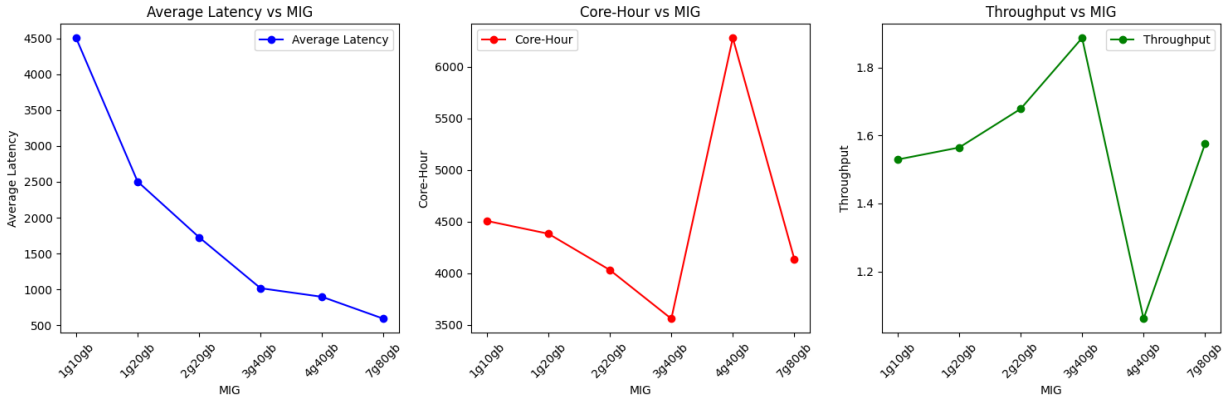

Fig. 4. Latency, execution cost (core-hours), and throughput for the BFS function executed with the V2a dataset on different MIG configurations

execution cost is lowest at 3g.40gb, right before the latency reduction begins facing diminishing returns.

B. Heterogeneous Workload

In the previous subsection, we evaluated the throughput of our serverless framework for homogeneous invocations of a single function on a single graph dataset. In this subsection, we look at the throughput of a trace (see Section IV-C for details) consisting of a heterogeneous mixture of functions and datasets to more accurately represent real execution scenarios. The composition of the heterogeneous workloads we use is shown in Table IV.

TABLE IV
COMPOSITION OF HETEROGENEOUS WORKLOADS

Workload	Functions	Datasets
1	BC, BFS, CC, PR, SSSP, SSWP	eu-2005 soc-sinaweibo patents
2	BC, BFS, CC, PR, SSSP, SSWP	ljournal-2008 indochina-2004 orkut-groupmemberships
3	BC, BFS, CC, PR, SSSP, SSWP	cage14 mawi-1 in-2004
4	BC, BFS, CC, PR, SSSP, SSWP	Stanford_Berkeley wiki-topcats kron_g500-logn21

Figure 6 shows how the throughput of our serverless framework varies for each heterogeneous workload based on the MIG configuration it executes on. The heatmaps show the throughput normalized for each heterogeneous workload because the absolute throughput depends on the specific choice of function and dataset. Our key observation from Figure 6 is similar to the homogeneous case: that the best MIG configuration depends on the workload.

However, in the case of heterogeneous workloads, another factor that impacts the best MIG configuration, besides latency, is the data transfer cost when multiple datasets are involved. We look at the impact of data transfer cost in the next section.

C. Benefit of Keeping Graph Datasets Warm

To highlight the benefit of keeping graph datasets warm, Figure 5 evaluates the latency and throughput of the serverless functions when keeping the datasets warm in both the CPU memory and the GPU memory or just the CPU memory. We do not show the performance of not keeping the datasets warm in the CPU memory and reading them from disk every time because it is too slow and unacceptable. The figure shows the performance of a heterogeneous workload with three functions and six datasets for different MIG configurations, particularly workload #1 from Table IV. The latency is broken down into data transfer time and compute time. The latency here refers to the total latency of all the functions if executed on one instance. We report this value instead of average latency because different functions in the heterogeneous workload have different latencies so their average is not a meaningful value.

From Figure 5, we make two key observations. The first observation is that compared to keeping the dataset warm in just the CPU memory, keeping the dataset warm in the GPU memory as well results in substantial latency reduction and throughput improvement due to a significant reduction in data transfer time. This observation shows the effectiveness of our graph dataset warming optimization. The second observation is that when keeping the dataset warm in both the CPU main memory and the GPU memory, as the MIG instance size increases, the transfer time decreases. This observation is due to the fact that larger MIG instance sizes have larger memory so they can support more graph datasets simultaneously, reducing how frequently a graph dataset gets evicted and needs to be transferred again when it is requested again.

This result showcases an important trade-off when using small or large MIG instances. As instances get larger, they may face diminishing efficiency, but they can accommodate keeping more datasets warm. This trade-off indicates why the heterogeneous workloads in Figure 6 favor larger instances compared to the homogeneous workloads in Figure 3.

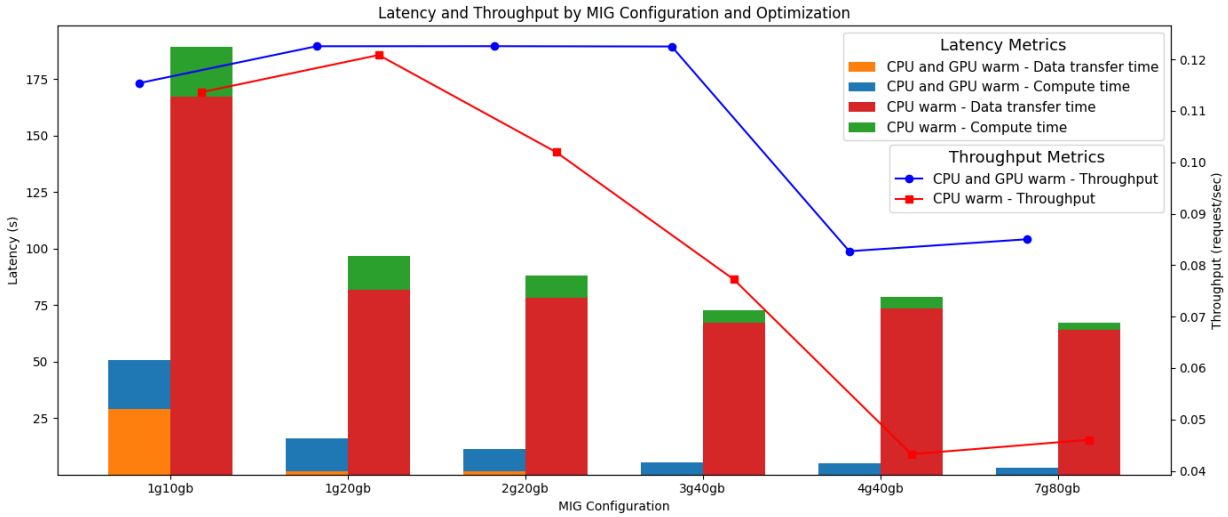

Fig. 5. Impact of keeping graph datasets warm on the latency and throughput of heterogeneous workload #1

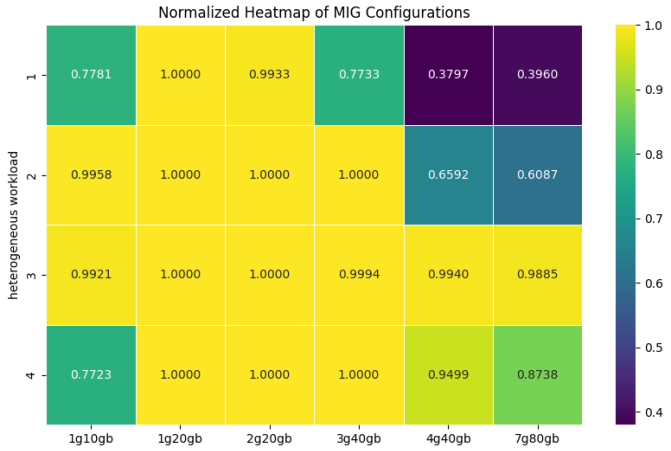

Fig. 6. Normalized throughput for different heterogeneous workloads across MIG configurations

VI. Conclusion and Future Work

We study the interaction between serverless computing and graph analytics on multi-instance GPUs. We use Knative as the serverless computing framework, Tigr for graph processing on GPUs, and the NVIDIA GPU Operator to enable GPU access in Kubernetes. Our evaluation shows that the MIG configuration that achieves the best throughput often depends on the function, dataset, and workload heterogeneity.

Based on these findings, our future work involves employing a strategy for dynamically reconfiguring the MIG instances to adapt to dynamically changing workload characteristics. To inform such a strategy, our future work also involves developing performance models for the serverless GPU kernels to assess their scalability across different MIG instance sizes when executed with different datasets. Moreover, while our evaluation has focused on homogeneous MIG configurations where all MIG instances have the same size, our future work also involves exploring heterogeneous MIG configurations where different MIG instances of different sizes may be specialized for different types of invocations. It also involves exploring MPS as an alternative to GPU sharing and comparing it with MIG.

References

[1] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[2] [Online]. Available: https://aws.amazon.com/lambda/

[3] [Online]. Available: https://azure.microsoft.com/en-us/products/functions

[4] [Online]. Available: https://cloud.google.com/functions?hl=en

[5] [Online]. Available: https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[6] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim, "Gpu enabled serverless computing framework," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2018, pp. 533–540.

[7] D. M. Naranjo, S. Risco, C. de Alfonso, A. Pérez, I. Blanquer, and G. Moltó, "Accelerated serverless computing based on gpu virtualization," *Journal of Parallel and Distributed Computing*, vol. 139, pp. 32–42, 2020.

[8] D. Du, Q. Liu, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Serverless computing on heterogeneous computers," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 797–813. [Online]. Available: https://doi.org/10.1145/3503222.3507732

[9] S. Werner and T. Schirmer, "Hardless: A generalized serverless compute architecture for hardware processing accelerators," 2022. [Online]. Available: https://arxiv.org/abs/2208.03192

[10] A. Garg, P. Kulkarni, U. Bellur, and S. Yenamandra, "Faaster: Accelerated functions-as-a-service with heterogeneous gpus," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2021, pp. 406–411.

[11] H. Fingler, Z. Zhu, E. Yoon, Z. Jia, E. Witchel, and C. J. Rossbach, "Dgsf: Disaggregated gpus for serverless functions," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 739–750.

[12] K. Satzke, I. E. Akkus, R. Chen, I. Rimac, M. Stein, A. Beck, P. Aditya, M. Vanga, and V. Hilt, "Efficient gpu sharing for serverless workflows," in *Proceedings of the 1st Workshop on High*

Performance Serverless Computing, ser. HiPS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 17–24. [Online]. Available: https://doi.org/10.1145/3452413.3464785

[13] H. Zhao, W. Cui, Q. Chen, S. Zhang, Z. Li, J. Leng, C. Li, D. Zeng, and M. Guo, "Towards fast setup and high throughput of gpu serverless computing," 2024. [Online]. Available: https://arxiv.org/abs/2404.14691

[14] S. Risco and G. Moltó, "Gpu-enabled serverless workflows for efficient multimedia processing," *Applied Sciences*, vol. 11, no. 4, p. 1438, 2021.

[15] T. Pfandzelter, A. Dhakal, E. Frachtenberg, S. R. Chalamalasetti, D. Emmot, N. Hogade, R. P. H. Enriquez, G. Rattihalli, D. Bermbach, and D. Milojicic, "Kernel-as-a-service: A serverless programming model for heterogeneous hardware accelerators," in *Proceedings of the 24th International Middleware Conference*, 2023, pp. 192–206.

[16] N. Pemberton, A. Zabreyko, Z. Ding, R. Katz, and J. Gonzalez, "Kernel-as-a-service: A serverless interface to gpus," *arXiv preprint arXiv:2212.08146*, 2022.

[17] H. Wu, Y. Yu, J. Deng, S. Ibrahim, S. Wu, H. Fan, Z. Cheng, and H. Jin, "{StreamBox}: A lightweight {GPU}{SandBox} for serverless inference workflow," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 59–73.

[18] [Online]. Available: https://spcl.inf.ethz.ch/Publications/.pdf/tobler-gpu-thesis.pdf

[19] M. Zhao, K. Jha, and S. Hong, "Gpu-enabled function-as-a-service for machine learning inference," 2023. [Online]. Available: https://arxiv.org/abs/2303.05601

[20] J. Gu, Y. Zhu, P. Wang, M. Chadha, and M. Gerndt, "Fast-gshare: Enabling efficient spatio-temporal gpu sharing in serverless computing for deep learning inference," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 635–644.

[21] A. H. Nodehi Sabet, J. Qiu, and Z. Zhao, "Tigr: Transforming irregular graphs for gpu-friendly graph processing," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 622–636. [Online]. Available: https://doi.org/10.1145/3173162.3173180

[22] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the gpu," in *Proceedings of the 21st ACM SIGPLAN symposium on principles and practice of parallel programming*, 2016, pp. 1–12.

[23] S. Kang, C. Hastings, J. Eaton, and B. Rees, "cugraph c++ primitives: vertex/edge-centric building blocks for parallel graph computing," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2023, pp. 226–229.

[24] R. Farahani, D. Kimovski, S. Ristov, A. Iosup, and R. Prodan, "Towards sustainable serverless processing of massive graphs on the computing continuum," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '23 Companion. New York, NY, USA: Association for Computing Machinery, 2023, p. 221–226. [Online]. Available: https://doi.org/10.1145/3578245.3585331

[25] L. Toader, A. Uta, A. Musaafir, and A. Iosup, "Graphless: Toward serverless graph processing," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2019, pp. 66–73.

[26] Y. Liu, S. Sun, Z. Li, Q. Chen, S. Gao, B. He, C. Li, and M. Guo, "Faasgraph: Enabling scalable, efficient, and cost-effective graph processing with serverless computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 385–400.

[27] R. Farahani, D. Kimovski, S. Ristov, A. Iosup, and R. Prodan, "Towards sustainable serverless processing of massive graphs on the computing continuum," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, 2023, pp. 221–226.

[28] R. Farahani, S. Ristov, and R. Prodan, "Designing a sustainable serverless graph processing tool on the computing continuum," in *European Conference on Parallel Processing*. Springer, 2023, pp. 210–214.

[29] [Online]. Available: https://knative.dev/

[30] [Online]. Available: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/index.html

[31] [Online]. Available: https://kubernetes.io/

[32] [Online]. Available: https://www.openfaas.com/

[33] A. Ebrahimi, M. Ghobaei-Arani, and H. Saboohi, "Cold start latency mitigation mechanisms in serverless computing: Taxonomy, review, and future directions," *Journal of Systems Architecture*, vol. 151, p. 103115, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762124000523

[34] M. Ghorbian, M. Ghobaei-Arani, and L. Esmaeili, "A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends," *Cluster Computing*, feb 2024. [Online]. Available: https://doi.org/10.1007/s10586-023-04264-8

[35] [Online]. Available: https://learn.microsoft.com/en-us/azure/container-apps/gpu-serverless-overview

[36] [Online]. Available: https://cloud.google.com/dataproc-serverless/docs/guides/gpus-serverless

[37] H. Hu, F. Liu, Q. Pei, Y. Yuan, Z. Xu, and L. Wang, "λgrapher: A resource-efficient serverless system for gnn serving through graph sharing," in *Proceedings of the ACM Web Conference 2024*, 2024, pp. 2826–2835.

[38] C. He, E. Ceyani, K. Balasubramanian, M. Annavaram, and S. Avestimehr, "Spreadgnn: Serverless multi-task federated learning for graph neural networks," *arXiv preprint arXiv:2106.02743*, 2021.

[39] V. M. Bhasi, A. Sharma, R. Jain, J. R. Gunasekaran, A. Pattnaik, M. T. Kandemir, and C. Das, "Towards slo-compliant and cost-effective serverless computing on emerging gpu architectures," in *Proceedings of the 25th International Middleware Conference*, 2024, pp. 211–224.

[40] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the gpu using cuda," in *International conference on high-performance computing*. Springer, 2007, pp. 197–208.

[41] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," *ACM Sigplan Notices*, vol. 47, no. 8, pp. 117–128, 2012.

[42] S. Diab, M. G. Olabi, and I. El Hajj, "Ktrussexplorer: Exploring the design space of k-truss decomposition optimizations on gpus," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–8.

[43] P. Yamout, K. Barada, A. Jaljuli, A. E. Mouawad, and I. El Hajj, "Parallel vertex cover algorithms on gpus," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 201–211.

[44] M. Almasri, I. E. Hajj, R. Nagi, J. Xiong, and W.-m. Hwu, "Parallel k-clique counting on gpus," in *Proceedings of the 36th ACM international conference on supercomputing*, 2022, pp. 1–14.

[45] M. Almasri, Y.-H. Chang, I. El Hajj, R. Nagi, J. Xiong, and W.-m. Hwu, "Parallelizing maximal clique enumeration on gpus," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2023, pp. 162–175.

[46] "Kourier: A lightweight knative networking layer," GitHub repository, available at https://github.com/knative/net-kourier.

[47] "Flannel: A network fabric for kubernetes," GitHub repository, available at https://github.com/flannel-io/flannel.

[48] [Online]. Available: https://crowcpp.org/master/

[49] Y. Sun, X. Gong, A. K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. McCardwell, A. Villegas, and D. Kaeli, "Hetero-mark, a benchmark suite for cpu-gpu collaborative computing," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–10.

[50] J. Gómez-Luna, I. El Hajj, L.-W. Chang, V. García-Floreszx, S. G. De Gonzalo, T. B. Jablin, A. J. Pena, and W.-m. Hwu, "Chai: Collaborative heterogeneous applications for integrated-architectures," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 43–54.

[51] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[52] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com

[53] [Online]. Available: https://graphchallenge.mit.edu/data-sets

[54] [Online]. Available: https://github.com/rakyll/hey

[55] [Online]. Available: https://github.com/sir-lab/data-release