Deep Learning Training on Multi-Instance GPUs

Anders Friis Kaas
IT University of Copenhagen
anfk@itu.dk

Ties Robroek
IT University of Copenhagen
titr@itu.dk

ABSTRACT
Deep learning training is an expensive process that extensively uses GPUs, but not all model training saturates the modern powerful GPUs. Multi-Instance GPU (MIG) is a new technology introduced by NVIDIA that can partition a GPU to better fit workloads that don’t require all the memory and compute resources of a full GPU. In this paper, we examine the performance of a MIG-enabled A100 GPU under deep learning workloads of three sizes focusing on image recognition training with ResNet models. We investigate the behavior of these workloads when running in isolation on a variety of MIG instances allowed by the GPU in addition to running them in parallel on homogeneous instances co-located on the same GPU.

Our results demonstrate that employing MIG can significantly improve the utilization of the GPU when the workload is too small to utilize the whole GPU in isolation. By training multiple small models in parallel, more work can be performed by the GPU per unit of time, despite the increase in time-per-epoch, leading to ∼3 times the throughput. In contrast, for medium and large-sized workloads, which already utilize the whole GPU well on their own, MIG only provides marginal performance improvements. Nevertheless, we observe that training models in parallel using separate MIG partitions does not exhibit interference underlining the value of having a functionality like MIG on modern GPUs.

1 INTRODUCTION
Deep learning models have defeated the world champion of Go [27], can write coherent news articles [6], and have surpassed human abilities in image recognition [12]. Although these types of models come at high computational costs, there are also deep neural networks that demand far less from the hardware [5, 13, 32]. Today, most deep neural networks are efficiently trained on Graphics Processing Units (GPUs) thanks to the embarrassingly parallel nature of their operations. However, if the training process does not fully saturate the resources provided by a GPU, the remaining computational power of the GPU goes to waste since the training process is given exclusive access to the GPU resources.

Workload co-location is a technique to increase hardware utilization when a workload does not require the entire compute or memory resources of a device. It refers to running multiple workloads simultaneously on the same device so that these workloads share the resources of that device. While workload co-location is heavily studied for CPUs [9, 11, 17], its opportunities and challenges have been largely unexplored for new generation GPUs.

In this paper, we examine the performance of a MIG-enabled A100 GPU under deep learning workloads of three sizes focusing on image recognition training. The three sizes represent different complexities and hardware resource needs of model training: (1) small with Resnet26 on CIFAR, (2) medium with Resnet50 on ImageNet64x64, and (3) large with Resnet152 on ImageNet. We investigate the behavior of these workloads when running in isolation on a variety of MIG instances with different compute and memory resources allowed by the A100 GPU in addition to running them in parallel on homogeneous MIG instances co-located on the same GPU. Our results demonstrate that:

- When model training is unable to utilize the full GPU on its own, i.e., the small case, training multiple models in parallel using several MIG instances has significant benefits. One can train seven ResNet26 models in parallel with a latency penalty of 2.5X over training the same model in isolation on the whole GPU leading to nearly 3 times the throughput. This can especially be beneficial for hyper-parameter tuning.
- In the medium and large cases the models are sufficiently large to saturate the GPU and co-location has marginal to no benefit. In addition, these models cannot run on the smallest GPU instances as the memory needs of the models exceed the memory on those instances (16GB).
- Co-located instances run in parallel without any interference as long as the available memory per instance is enough for the runs, and co-located and isolated runs over the same type of instances perform similarly.

Rest of the paper is organized as follows. Firstly, Section 2 gives background on MIG and surveys related work. Section 3 follows this up with our experimental methodology and setup. Finally, Section 4 discusses the results and Section 5 and Section 6 draw conclusions from our findings.

2 BACKGROUND AND RELATED WORK
This section first provides some background on MIG and how to create MIG partitions. Then, we survey related work on benchmarking deep learning and MIG.

2.1 Multi-Instance GPU (MIG)
Multi-Instance GPU (MIG) is a recent technology bundled with NVIDIA’s Ampere GPUs. It allows these GPUs to be split into
was released in 2016 and contains eight workloads, each of which involves training of a distinct (at the time) state-of-the-art deep learning model. 2017 saw the introduction of BenchIP [29], which contains a combination of microbenchmarks and macrobenchmarks based on the size of the workload.

2.2.1 Benchmarking with deep learning. There has been several works. Most TPCx-AI use cases, on the other hand, do not sufficiently stress the GPU hardware. Therefore, in this work, we design a custom benchmark (see Section 3.3) to both stress the GPUs under test and scale the workload up and down.

2.2.2 MIG. MIG is a relatively new technology and there has not been many works that thoroughly explore its possibilities. Wang et al. [32] compare the effectiveness of their hardware utilization squeezer to MIG when it comes to improving both the GPU utilization and deep learning training times. Tan et al. [28] build a system that, given a set of deep learning inference tasks and service-level objective constraints, is capable of automatically and seamlessly reconfigure MIG-enabled GPUs on Amazon Web Services (AWS) to the most efficient MIG profile configuration. Our work is complementary to these works since we focus on devising an experimental methodology to investigate the strengths and limitations of MIG.

3 METHODOLOGY

This section provides a detailed overview of our methodology. First, Section 3.1 describes the hardware used for conducting our experiments. Next, Section 3.2 defines our metrics, their relevance, and how we measure them, followed by the workloads we designed for this study in Section 3.3. Finally, Section 3.4 details the list of the experiments ran.

3.1 System

For characterizing the performance of MIG, we use a DGX Station A100, which is composed of an AMD EPYC 7742 CPU and four A100 GPUs.
The CPU consists of 64 cores, amounting to 128 logical cores (threads) operating at a base clock of 2.25 GHz and capable of reaching a maximum boosted clock of up to 3.4GHz [3]. The L3 cache is 256MB and DRAM is 512GB. The A100 GPUs have 40GB of high bandwidth memory, and the graphics processors are based on the SXM form factor [22]. They support a maximum of 7 MIG instances at 5 GB of memory per instance (see Section 2.1). The DGX Station A100 setup represents a pre-packaged solution provided by NVIDIA, and the operating system is DGX OS, which is a variant of Ubuntu 20.04.4 LTS with hardware-specific optimizations.

3.2 Metrics

The metrics we used to reason about the performance of MIG can be classified into three categories: application-level metrics (Section 3.2.1), GPU metrics (Section 3.2.2) and CPU metrics (Section 3.2.3).

3.2.1 Application-level metrics. Application-level metrics are the metrics related to model training.

Time per epoch is the time it takes to finish a single epoch of training for a particular model. The reason GPUs are used in deep learning is to reduce training time by exploiting the embarrassingly parallel nature of most deep learning computations. Therefore, time per epoch is arguably the most fundamental metric to look at and optimize. We obtain the time per epoch by dividing training time by the total number of epochs we use to train the models.

Accuracy indicates the predictive power of a machine learning model. For a set of categorical predictions, it is defined as the share of predictions that are correct. While accuracy does not determine how well hardware resources are utilized, and hence not the primary metric in this work, one should never ignore it when focusing on machine learning. We record the training and validation accuracy to ensure the models are training correctly.

3.2.2 GPU metrics. Throughout our experiments, the GPU metrics are reported in the context of both the full GPU and its respective partitions. While instances are separate units that execute each workload with the provided number of resources, they are also part of the full GPU. In addition, the full GPU has an extra compute unit that cannot be included as part of the instances ([23] and Section 2.1). Therefore, it is essential to track these metrics described below both at the instance-level and at the level of the whole GPU.

Except for the GPU memory metric, which is collected using the NVIDIA System Management Interface (nvidia-smi), we use the Data Center GPU Manager (dcgm) to collect all the GPU metrics. nvidia-smi does not provide measurements with MIG instances and dcgm does not measure GPU memory used. Therefore, we need both of these sources to fetch all the required information.

GRACT, Graphics Engine Activity, shows the fraction of time any portion of the graphics or compute engines were active. We are particularly interested in observing how active (and utilized) the whole GPU and its respective instances are depending on the workload being executed.

SMOCC, SM Occupancy, is the fraction of resident warps on a multiprocessor, relative to the maximum number of concurrent warps supported on a multiprocessor. It is a complementary metric to SMACT and SMOCC. High SMOCC indicates more effective GPU usage for workloads that stress the memory or memory bandwidth. However, if the workload is compute-bound, high SMOCC may not necessarily indicate more effective GPU usage.

DRAMA, Memory Bandwidth Utilization, is the fraction of cycles where data was sent to or received from device memory. Higher DRAMA shows higher memory utilization of the device, and serves as a complementary metric to the three metrics above that focus on the utilization of a GPU’s compute resources.

GPU memory usage for models being trained is essential for deciding the ideal setup for MIG partitioning as the partitions can have varying amounts of memory. By default, a deep learning framework like TensorFlow allocates all available GPU memory from the moment training starts, which is done to reduce memory fragmentation [30]. However, this prevents us from monitoring the actual GPU memory consumption characteristics of our training workloads. Therefore, we disable such behavior during our experiments to quantify the GPU memory actually required by the workload.

3.2.3 CPU metrics. In addition to tracking the GPU compute and memory utilization, we also monitor how MIG impacts the CPU usage characteristics.

CPU utilization We monitor CPU utilization on the process level as an aggregate over the threads of the training process using the tool top. Since the DGX Station has 128 logical cores, the maximum utilization of the CPU would be 128 × 100% = 12,800%. 100% CPU utilization could therefore mean a few different things. It could mean that one of the logical cores is working at maximum capacity. Alternatively, it could mean that n cores are working with an average of 1/n utilization per core.

Main memory usage on CPU is also an important metric to monitor. Some operations, such as model initialization and data management, depend on the CPU memory. We monitor total memory allocation to training processes in order to observe the memory requirements of running a single or multiple models using top. More specifically, we report RES, resident memory, which is the total physical memory allocated to a process.

3.3 Workloads

We design three workloads of different sizes (small, medium, and large) to assess the performance of MIG under different loads. All three workloads consist of training a ResNet model on an image dataset. The complexity of the dataset and model vary between workload sizes. Here, we first go over the selected datasets after
which we discuss the models. All the workloads are implemented in TensorFlow [1].

### 3.3.1 Data and preprocessing

The three workloads have different dataset sizes.

The **small** workload, with `resnet_small`, is trained on CIFAR-10 [14], which is a relatively small dataset containing 60,000 labeled 32 × 32 pixel images divided over 10 classes. The dataset is split into 50,000 training images and 10,000 test images. The entire CIFAR-10 dataset uses approximately 32 × 32 pixels × 3 channels × 8 bytes × 60,000 images ≈ 1.5 GB of memory. It is therefore feasible to load the entire dataset into memory at runtime instead of dynamically streaming the data into memory from disk. In terms of preprocessing, we normalize the dataset by subtracting the mean image from every image in the dataset as suggested in [10]. We train `resnet_small` on 90% of the training set and use the remaining 10% as a validation set.

The **medium** workload, with `resnet_medium`, uses a downsampled version of the ImageNet2012 dataset called ImageNet64×64 [8]. ImageNet2012 is a collection of 1,431,167 labeled images from 1,000 different classes. The dataset is split into 1,281,167 training images, 50,000 validation images, and 100,000 test images. Unlike CIFAR-10, the dataset is not balanced and the images of ImageNet are furthermore not all uniform in size.

While loading the whole scaled-down dataset into memory would in theory be possible, as it only demands 64 × 64 pixels × 3 channels × 8 bytes × 1431167 images ≈ 17.5 GB, we decided against doing this. The main reason is that we wanted to make the medium-sized workload comparable to the large workload so that the only differences between these two experiments would be the size of the dataset and the size of the model. Secondly, some of our experiments involve training multiple models in parallel which would require loading up to seven versions of the dataset into memory, which could present an issue. Instead of loading all of the images into memory, we use the data generator `ImageDataGenerator` from TensorFlow to dynamically stream training data from disk. The images are also preprocessed using the `imagenet_utils.preprocess_input` function, and we empirically determine the smallest optimal values of `workers` and `max_queue_size` to be 1 and 10, respectively. Setting the number of workers to 1 means that TensorFlow will create and use 1 CPU thread to fetch training data and setting the maximum queue size to 10 means that a maximum of 10 preprocessed batches of training data will be stored in RAM at any point in time. The purpose of storing more than one batch of training data in RAM at the same time is to try to minimize the amount of time that the GPU is spending on waiting for training data. To determine the optimal values for `workers` and `max_queue_size`, we used Tensorboard, which shows the amount of time spent on waiting for input to be fed to the model. The values of `workers` and `max_queue_size` were then gradually increased until the time spent on input was close to 0.

The **large** workload, with `resnet_large`, is trained on ImageNet2012 [25]. Every picture is resized to 224 × 224 using the nearest pixel interpolation method in order to conform with the size of images used in the original ResNet specification [10]. Loading all of ImageNet into memory is impossible due to its exceedingly large size (greater than 100GB). Instead, we dynamically load batches into memory once again using the data generator `ImageDataGenerator` that is available in TensorFlow. This generator automatically fetches a batch of training data at a time from disk, preprocesses it and stores it in RAM, and then transfers it to GPU memory where it can be consumed by the ResNet model. Additionally, we experimentally determine `workers=16` and `max_queue_size=20`, using the same methodology to the medium case for these parameters.

### 3.3.2 Deep learning models

All three workloads feature a ResNet convolutional network as their model [10]. ResNets are a very popular choice of model for image classification and image segmentation [16, 26] and are easy to scale up and down in size. This aligns well with our use case where we want to test the performance of GPU MIG instances of different sizes. We train a ResNet26V2, ResNet50V2, and ResNet152V2 models for the small, medium, and large workload cases, respectively. The larger ResNet models have more layers and parameters. The medium model has about twice the number of parameters as the small one, and the large model has about twice the number of the medium model.

### 3.4 Experiments

This subsection provides an overview of our experimental runs. Each experiment was run twice (i.e., replicated) to ensure reliable results.

We use a batch size of 32 for all the models to strike a balance between statistical accuracy, memory requirements, and time per epoch. All experiments with the small model train for 30 epochs, while all medium and large model runs are for 5 epochs. This was done to strike a balance between training results and time spent on a run, since there is a dramatic increase in complexity and time to completion for medium and large model runs.

The number of concurrently trained models depends on the MIG profile used, which determines the allocation of GPU SM and memory slices. All of our experiments cover workload executions on a non-MIG instance and the five MIG profiles provided by NVIDIA (Section 2.1. For each profile, and for each dataset size, we run two types of runs. The first runs one training in isolation on an instance of that profile. The second runs several homogeneous MIG instances in parallel all training a model at the same time. We examine the maximum possible instances that can be configured for a given profile. For example, for the 1g. 5gb profile there is respectively one training run on a single instance of the profile and a second run with the maximum possible instances that can be generated for that profile, which is 7, where 7 models are trained in parallel. Furthermore, we decided to focus on homogeneous instances for the parallel runs in this study to scope down the total number of experiments. Investigation of the impact of different instance combinations are left for future work.

There are two exceptions to the parallel runs using MIG profiles. This is due to the 7g. 40gb and 4g. 20gb profiles not being able to have existing parallel instances due to the maximum available resources. Furthermore, we do not report GPU metrics derived from DCGM for 4g. 20gb due to DCGM not reporting anything for this profile. Section 5 further elaborates this issue. However, in the case of a single instance run, we deem an experiment with 3g. 20gb profile comparable to 4g. 20gb. The experiment with the full MIG profile, 7g. 40gb, has the purpose to explore possible alterations of
We now discuss and analyse our results. In total, a full run of our software and drivers. The versions of the nvidia-smi and GPU drivers for the A100 GPUs, are 510.47.03. The CUDA version used is 11.6. DCGM is at version 2.3.2. In addition, we use the Python version 3.9.7, while TensorFlow is at version 2.7.0.

4 RESULTS

We now discuss and analyse our results. In total, a full run of our experiments took approximately 135 hours or about five and a half days. Running the medium and large workloads on the smallest GPU instance, 1g.5gb, resulted in an out-of-memory error and we therefore only have full run results to show for resnet_small in the 1g.5gb experiments.

Throughout the reporting of DCGM data, the tool was unexpectedly terminated on two occasions, resulting in only partially complete data for our initial experiments. These were the setups for the non-MIG-enabled GPU and 3g.20gb one profile, impacting only our large workload executions. In our efforts of carrying out replications of the experiments, we identified that the data in said replications can be beneficial to supplementing the current analysis. While there were another two occasions of such occurrence in the replication data, they were for different instances and our original data was not impacted. We had access to data resulting from an alternative experiment with identical setup for all of our workload executions. Therefore, the complete data for large workload execution on 3g.20gb one and non-MIG setups has been included in the visualizations we present in the upcoming section. While the incomplete data was highly similar to the complete results from the replications, we discuss these challenges in section 5. Furthermore, metrics reporting for the 4g.20gb instance are not viable due to challenges with querying metrics from DCGM for this instance size.

Overall, the results indicate that smaller GPU instances result in longer training times but increased utilization. Additionally, we found no penalty in training several models in parallel in different GPU instances compared to training a single model at a time. Lastly, for the medium and large workloads, we do observe a significant difference in using the A100 in non-MIG mode and using it in MIG mode as a 7g.40gb GPU instance.

4.1 Time per epoch

Figure 2 shows the epoch times for resnet_small. Each bar represents the total amount of time needed to complete one epoch of training for resnet_small averaged over 30 epochs of training. For the experiments where we trained multiple models in parallel, the bars have been given the same color, e.g. the brown bars with a star hatch pattern represent seven models trained in parallel.

From the chart it can be seen that, in general, using smaller instances results in longer training times. However, the relationship between number of compute slices and training time is not 1:1. For instance, training an instance of resnet_small on a 1g.5gb instance takes approximately 39.8/16.1 = 2.47 times longer than training it on the 7g.40gb instance despite having 1/7 as much compute power and 1/8 as much memory.

We also observe that training multiple models in parallel on separate instances has no significant effect on the training time when compared to training a single model on an instance of the same size. For example, training a single instance of resnet_small on a 2g.10gb instance for one epoch took 25.7 seconds whereas training three models in parallel on 2g.10gb instances took between 25.6 and 26.0 seconds for one epoch. This supports the claim by NVIDIA that separate MIG GPU instances are completely isolated from each other.

The fact that there is not a 1:1 relationship between the size of the GPU instance and the training time combined with the fact that there is no training time penalty associated with training multiple models in parallel presents a unique opportunity with MIG. When performing hyperparameter optimization of a machine learning model, one could run seven models in parallel with different hyperparameter settings on seven different instances of 1g.5gb. This would be significantly faster than sequentially running the model seven times on a 7g.40gb instance. As an example, for resnet_small, it would take (7 × 16.1)/39.8 = 2.83 times as long to train seven models sequentially on a 7g.40gb instance than in parallel on seven 1g.5gb instances.
Charts showing the time per epoch for the large and medium models can be seen in fig. 3. The processes running the medium and large workloads crashed immediately when running on 1g. 5gb. For the experiments that ran correctly we saw a much larger penalty associated with running the workload on smaller instances. For instance, running one epoch of resnet_medium in a 7g. 40gb instance took 35.4 minutes, whereas running three workloads in parallel in 2g. 10gb instances took 106.8 minutes per epoch. Running three medium workloads sequentially in a 7g. 40gb instance thus takes almost exactly the same time as running three medium workloads in parallel in 2g. 10gb instances \((35.4 \cdot 3)/106.4 = 0.99\). For resnet_medium we saw marginal improvements in running two parallel workloads in 3g. 20gb instances compared to running them sequentially in a single 7g. 40gb instance. For resnet_large we saw very similar results as for resnet_medium; running three workloads in parallel in 2g. 10gb instances took exactly as long as it would have taken to run three large workloads sequentially in a 7g. 40gb instance.

Regarding the non-MIG experiments, we see slightly faster executions in the non-MIG runs compared to the 7g. 40gb runs for all resnet_small, resnet_medium and resnet_large. The improvement is smallest for resnet_small where the non-MIG time per epoch is only 0.7% faster than the 7g. 40gb time per epoch. For resnet_medium and resnet_large, however, we clearly see significant improvements when disabling MIG. The time per epoch for a resnet_medium run in a non-MIG GPU is 2.8% faster than in 7g. 40gb. For resnet_large the improvement is even larger at 2.9%. A likely reason why the non-MIG runs were slightly faster is that a computer unit gets disabled when MIG is enabled. The small workload might not have benefitted as much from disabling MIG as it already did not fully utilize all of the compute power available in 7g. 40gb. Providing it with even more SMs would therefore not be beneficial.

### 4.2 GPU utilization

We proceed by going over the DCGM and GPU memory metrics.

#### 4.2.1 DCGM

We provide graphic representations of the performance of 4 metrics obtained from DCGM. Our focus is on Graphics Engine Activity (GRACT), Streaming Multiprocessor Activity (SMACT) and Occupancy (SMOCC). In addition, we provide a brief analysis of the performance of Memory Bandwidth Utilization (DRAMA). For each of the 3 workloads, we created two types of graphs per metric. The first type focuses on the metric performance for the full GPU, while in the second we display information for the individual instances to better understand their impact. Separately and as a baseline, we also provide DCGM metrics for a full GPU with MIG mode disabled. An expectation was that the results reported from these model runs will be similar, or nearly identical, to the ones reported by the 7g. 40gb instance.

On the y-axes we display the median of the reported average-over-time metric values in percentages. Our x-axes show the different device groups corresponding to the aforementioned MIG profiles. In a 3g. 20gb parallel example, consisting of two parallel 3g. 20gb instances as part of the whole GPU, the first graph shows the median value of a given metric for the full GPU while the second one displays the median of that metric for each of the two instances. It is worth remembering that our device allocations are homogeneous, and this is why we omit the otherwise possible 1g. 5gb instance in this allocation. In terms of non-MIG reporting, we include the same device-level values in both device- and instance-level visualizations. This is a decision taken to allow for comparison with a baseline, both for the full device but also across instances.

In the following analysis, we will look at specific device groups’ performance and their comparison to each other, for each metric. Our explicit focus is on the MIG-enabled device groups. Towards the end of each metric analysis, we specifically focus on a comparison with the case for a non-MIG device.

**GRACT.** The highest reported Graphics Engine Activity for the small workloads results from the 1g. 5gb parallel device group. The reported activity across its seven instances ranges from 90.2% to 90.5%, which amounts to an average of 90.2% utilization of the full GPU. Similar result is produced by the device group covering an individual instance of the 1g. 5gb profile where the overall device activity is dramatically lower due to the small fraction of overall resource utilized (see fig. 4a and fig. 4d).

Although the 2g. 10gb parallel device group reports \(\approx 94\%\) activity throughout its instances, the overall device activity is at 71.8% for the small workload. The 2g. 10gb profile is the second-to-highest in terms of GRACT both for individual and parallel instance allocations. The lower overall device utilization, in comparison to the 1g. 5gb parallel group, can be explained by the omitted compute slice. This is due to the homogeneous nature of device groups used in our experiments. If an additional 1g. 5gb allocation was utilized in parallel, which is supported by NVIDIA, a higher value would be expected.

Similar is the case with 3g. 20gb parallel, where one compute slice remains unused, despite the complete allocation of available memory. 3g. 20gb one, however, is \(\approx 10\%\) points more utilized than 2g. 10gb one when it comes to the entire device - despite the instance-level utilization being 5.2 percentage points lower. A possible explanation is the previously mentioned consideration for allocated resources per instance, with respect to the total available for the device. More allocation does not necessitate better utilization in terms of instance-level activity.

An interesting detail is the activity of the partition covering the entirety of the GPU resource as part of the 7g. 40gb profile. Using that profile, the reported utilization is at 71.6% for the small model and is the lowest of all reported. Such result could entail that for workloads of that size, a GPU with such specifications is better utilized if smaller portions of it are used as an alternative. This comes at the cost of extra time to completion, which we discussed separately. However, once the workloads grow in scale, the differences between device groups’ GRACT performance diminish.

In the medium workload (see fig. 4b and fig. 4c), the 2g. 10gb one group reported highest instance-level activity at 96.3%. The 2g. 10gb parallel group reported highly similar values across its instances at 96.1% and second-highest utilization of the full device - 82.4%. The highest device utilization was achieved by the 7g. 40gb one profile at 88.6%. While its respective instance is the lowest utilized in comparison to the rest of the profiles’ instances – also
Deep Learning Training on Multi-Instance GPUs

In the large workload (see fig. 4c and fig. 4f), the highest utilized instances belonged to the 2g.10gb profile, reporting identical values of 96.9%. The difference between the highest- and lowest-performing profiles in the large run is 6.1 percentage points. This is a display of the shrinkage across setup utilization differences throughout scaled workloads.

There was an interesting phenomenon when it comes to full device utilization. In the small run, the 7g.40gb one profile was utilized at 71.6% while 1g.5gb parallel was most utilized reporting 90.2% - a difference of ≈19 percentage points. In the medium workload, the 7g.40gb profile reporting 88.6% utilization exceeded all other profiles with a device-level difference of 6.2 percentage points compared to the second highest-utilized device - the 2g.10gb parallel. Similar behavior was observed in the large workload, where the difference between the two highest-utilized devices was 7.7 percentage points.

However, as we acknowledged, the context of full device metric calculations can turn out to be misleading and - in our case - incomplete. We are only using homogeneous instances, sometimes resulting in unutilized resources. In the very case of 2g.10gb parallel, an extra 1g.5gb instance might have lowered the difference between the two device groups, or possibly changed the order of ranking.

In terms of non-MIG device data, the reported GRACT for all workloads was highly similar to the values for 7g.40gb. In all cases, the 7g.40gb profile reported slightly higher activity, not surpassing 0.2 percentage points.

SMACT. When it comes to the activity of streaming multiprocessors, the highest instance-level metric values for the small workload were reported by the two 1g.5gb device groups. The performance of the instances in the parallel device group ranged from 75.2% to 88.6%, the difference from the highest-utilized one, 2g.10gb one, is 7.7 percentage points.

In the large workload (see fig. 5c and fig. 5f), the highest utilized instances belonged to the 2g.10gb profile, reporting identical values of 96.9%. The difference between the highest- and lowest-performing profiles in the large run is 6.1 percentage points. This is a display of the shrinkage across setup utilization differences throughout scaled workloads.

However, as we acknowledged, the context of full device metric calculations can turn out to be misleading and - in our case - incomplete. We are only using homogeneous instances, sometimes resulting in unutilized resources. In the very case of 2g.10gb parallel, an extra 1g.5gb instance might have lowered the difference between the two device groups, or possibly changed the order of ranking.

In terms of non-MIG device data, the reported GRACT for all workloads was highly similar to the values for 7g.40gb. In all cases, the 7g.40gb profile reported slightly higher activity, not surpassing 0.2 percentage points.
overall instance SM Activity by \(\approx 14\) percentage points. However, as we discussed earlier, such increases may benefit the time to completion. The lowest-utilized instance was from the 7g. 40gb one profile with 40% reported SM Activity.

Drawing on these results, it can be argued that the 40GB version of A100 is not well utilized when it comes to our small workload experiment - especially if the full device is used. If 7 instances are used, it would mean that the total amount of work completed is 7 times more compared to the case of a single instance. This can be beneficial depending on the context and use case - despite the reported device-level SM Activity of 75.1% being less than the suggested 80% as an indicator of effective utilization. As we covered earlier in section 3.2, all values less than 50% can be indicative of ineffective usage, while a categorization for effective usage would require more than 80%. Three of our profiles, 3g. 20gb, 2g. 10gb, 1g. 5gb, and their instance device groups reported values higher than 50% but less than 80% which we argue to be in the neutral range. However, the reported \(\approx 75\%\) SMACT for 1g. 5gb is close to the suggested value indicative of effective utilization. The 7g. 40gb profile and its instance device group can be categorized as an ineffective choice.

It is however in the medium and large workloads (respectively fig. 5d and fig. 5f) when the results for SM Activity become more interesting. For both of these workloads the reported SM Activity values across device groups not only follow the same pattern, but are almost the same between the two workloads - with insignificant differences. This is seen in the metric reporting for instances, as well as for the full device (see fig. 5c and fig. 5e).

In both medium and large workload executions, the least SM Activity per instance is reported by the 7g. 40gb one device group. However, this time the respective values are 73.4% and 74.4%. This represents a growth of more than 30 percentage points when compared to the performance of the same device group in the small workload execution. 73.4% and 74.4% are both values that are close to the suggested 80%+ for a classification of a device as effective. An explanation for the higher activity can be the nature of the workloads expressed in their larger scale - which has impact on both the utilization of compute units and memory. We discuss memory in more detail in the upcoming analysis of the SM Occupancy results, as well as in a dedicated subsection as part of this analysis.

The highest value for SM activity, also in both medium and large workloads, results from the 2g. 10gb instances. It is within the range of 91.3% and 91.8% across the two workloads, which represents a minor difference. In the medium workload, the 2g. 10gb one device group slightly outperforms its parallel analogue on instance-level by \(\approx 0.2\) percentage points. In the large workload, one of the parallel instances matches the SM Activity value of the individual profile’s instance, while the rest report highly similar results. The range, therefore, between the least and highest-reported SM Activity for medium and large workload runs, is \(\approx 18\) percentage points. However, if we reduce the context for calculating the ranges to only include partitioned device groups (i.e., excluding the 7g. 40gb one profile), that range is decreased to \(\approx 6.5\) percentage points.

Overall, for both types of workload runs, 4 out of 5 device groups, respectively belonging to 2 distinct profiles, can be classified as effectively utilized when it comes to SM activity per instance.

In the context of the full device SM activity reporting, 7g. 40gb one slightly outperforms 3g. 20gb parallel in both medium and large workload runs. The 2g. 10gb parallel device group is still most effective with \(\approx 78.5\%\) SM Activity. None of the device groups reported values higher than 80% for the full device. However, it is once again worth acknowledging the fact that the device groups we use for all experiments are homogeneous and do not sum up to the entirety of the device resource. This is also reflected in the device-level calculations.

In consideration of the non-MIG-enabled device, all reported SMACT values were once again highly similar to the 7g. 40gb profile, with minor differences in benefit to 7g. 40gb as part of the small workload execution.

**Figure 6:** Median SMOCC for all experiments.

**SMOCC.** The small workload (fig. 6a and fig. 6b) reports the lowest SM Occupancy values from all 3 workload types. Across the instance device groups, the 1g. 5gb profile reports the highest values ranging between 34.9% and 35.4%, while the lowest value is once again resulting from the device group for the 7g. 40gb one instance with 20.3%. The device-level values are highest for the 1g. 5gb parallel device group (35%), while 7g. 40gb one reports the same value as its instance - 20.3%.

In consideration with the SMACT values for this workload reporting lower activity, the low SMOCC values further enhance the hypothesis that the GPU can be deemed underutilized for such type of workload. Another relevant aspect is the reported Graphics Engine Activity. While it is relatively high for the small workload across most of the instances, there are still significant differences in comparison to the GRACT for medium and large workloads. These differences often result in \(\approx 15\) percentage points or more in benefit to the medium and large workloads’ reported values.
For medium and large workload executions (as seen in fig. 6c, fig. 6d, fig. 6e and fig. 6f), the distribution of values for SM Occupancy follows a similar trend to the one for SM Activity. In addition, each of the device groups in the medium workload execution shares almost identical SMOCC values to its respective analogue in the large workload. This is observable both across the metric reporting for the full device, as well as the individual instances. For both workload types, the highest occupancy across instances results from the 2g.10gb profile and its respective device groups, while the lowest is part of the 7g.40gb one partition. The difference between lowest- and highest-reported occupancy per instance is 17.7 percentage points for both workloads.

Drawing on these results, it is also worth considering the effect of memory bandwidth limitations[21]. Our workloads are not strictly bandwidth-limited when it comes to memory. However, the medium and large workloads can be considered both compute- and memory- limited due to their larger scale. Referring back to the Graphics Engine Activity, we see values in the ≈90% range for most of the instances used across these workloads, while for SM activity most values reported are near the ≈85% range. This can be considered as high utilization with relatively high (mostly 50%-60%) SM Occupancy.

We acknowledge the complex nature of calculating Occupancy and implications of categorizing reported values for that metric as effective or ineffective. As reported in [18], "Low occupancy results in poor instruction issue efficiency, because there are not enough eligible warps to hide latency between dependent instructions". Conversely, "When occupancy is at a sufficient level to hide latency, increasing it further may degrade performance due to the reduction in resources per thread" [18]. We acknowledged the levels of achieved occupancy across our experiments and analyzed them in a context supported by the workload memory requirements, limitations of some of the profiles, as well as reported GRACT and SMACT. However, our analysis is not comprehensive when it comes to a detailed examination at the occupancy for the type of workloads we execute. This is due to the complex nature of this metric and the many additional factors that need to be taken in consideration for a comprehensive study. We consider that further and specialized efforts in the area could be more beneficial. In terms of graphics memory, our attempts to execute the medium and large workloads on a 1g.5gb instance have failed due to the memory limitations of that profile. When it comes to compute limitations, the main trade-off we identified was time to completion against amount of completed work. While we recognize the possibility of compute slices being a bottleneck in a case where more memory is allocated, examining such instances was also not at the core of this project’s scope.

In all cases, the non-MIG GPU also shares highly similar SMOCC values to the 7g.40gb profile. The highest difference is reported in the small workload, 1.4 percentage points. For the medium and large workloads, the differences are less than 0.5 percentage points. This could indicate that, typically, the non-MIG GPU performs in a very similar manner to a MIG instance utilizing the full device’s resource.

**DRAMA.** When it comes to the memory bandwidth utilization (DRAMA), the instance-level value for each device group was highly similar and almost identical to its respective analogue across all 3 workloads. The highest-reported values resulted from the 2g.10gb profile, followed by 3g.20gb and 7g.40gb (see fig. 7).

The values reported on device-level were lowest for the small workload, ranging between 3.5% and 24.8% - respectively for the 1g.5gb one and 1g.5gb parallel device groups. Therefore, for this workload, the highest device-level memory bandwidth utilization was reported by the 1g.5gb parallel device group. While the lower reported value for 1g.5gb one can be explained by the context that is used for device-level calculations, 7g.40gb one reported a similarly low value of 6.1%.

When it comes to the performance in medium and large workloads, we observed a trend between the two, expressed in similar device-level values across device groups’ respective analogues in the two workloads. Another difference was seen in the overall higher reported values in comparison to the small workload. In both medium and large workloads, the highest reported device-level value was for the 3g.20gb parallel instances (=52%), followed by 2g.10gb parallel (=49%) and 7g.40gb one (=44%). As expected, the profiles for 2g.10gb and 3g.20gb covering a single instance reported much lower device-level values. This can be explained by the fact that only a single instance is used.

On device-level, the reported values for the non-MIG-enabled device were highly similar to the ones for 7g.40gb

![Figure 7: Median DRAMA for all experiments.](image-url)

4.2.2 GPU memory. Figure 8a shows the maximum amount of allocated GPU memory per experiment. Maximum amount may be misleading since these amounts were allocated at the very beginning of each experiment and did not fluctuate during the whole run. The first thing to notice in the graph is that there is no difference
in the amount of allocated memory between using the GPU in non-MIG mode and using a 7g.40gb instance. Furthermore, it can be seen that given optimal conditions (i.e. non-MIG mode or 7g.40gb where there is 40 GB of available memory), resnet\_small uses 9.5 GB, resnet\_medium uses 10.4 GB and resnet\_large uses 19.0 GB. This is also the case in both the 4g.20gb one and the 3g.20gb one experiments. These amounts of memory seem to be what Tensorflow considers the optimal amounts of memory to use given the models and the training data.

However, it can be seen from the chart that the models are still able to train when less memory than they ‘prefer’ is available. For instance, in the 2g.10gb one experiment, resnet\_large uses only 9.9 GB of memory, which is about half of its memory consumption in the 7g.40gb one experiment, and in the 1g.5gb one experiment, resnet\_small is able to train using only 4.7 GB of memory. This indicates that Tensorflow is aware of its hardware environment and is able to adapt the training process to the amount of available memory.

Lastly, it can be observed that within a particular MIG profile, training \( n \) models in parallel simply uses \( n \) times as much GPU memory as training a single model. For example, training two resnet\_medium models in the 3g.20gb parallel experiment uses two times as much memory as training a single resnet\_medium model in the 3g.20gb one experiment.

### 4.3 CPU and main memory

In addition to the GPU-related metrics, we also monitored main memory consumption and CPU utilization. Here we present the main memory results (section 4.3.1) and the CPU utilization results (section 4.3.2). Overall, our results indicate that both main memory consumption and CPU utilization is proportional to the number of models training in parallel on the GPU.

#### 4.3.1 Main memory consumption

Figure 8b shows the maximum aggregate amount of physical RAM allocated to the process(es) running our workloads for each experiment as calculated using the procedure outlined in section 3.2.3.

The bars in fig. 8b represent the amount of RAM needed to be able to run each experiment. We see that running a single resnet\_small workload requires approximately 7.1 GB of main memory and running a single resnet\_medium workload demands only 5.4 GB. The reason why resnet\_small has a larger memory footprint than resnet\_medium is likely that resnet\_small stores all of its training data in memory. On the other hand, resnet\_medium streams the training data from disk dynamically, thereby requiring a smaller
working set. Lastly we see that resnet_large requires at most 12.6 GB of memory.

From the figure it can be observed that the GPU instance size does not significantly impact the maximum main memory requirements, e.g. resnet_small takes up about 7 GB of memory in all of the non-parallel experiments. One exception to this is the 4g.20gb one run where resnet_large has a maximum resident memory requirement of 12.6 GB which is more than 2 GB more than for the other non-parallel experiments. We also saw this difference in our experiment replication results and we are not sure what causes it.

Lastly, the graph shows that training \( n \) models in parallel requires about \( n \) times as much memory as training a single model. This implies that a user must also have a fairly large amount of available main memory in order to reap all the benefits of MIG. For example, running a single resnet_small workload uses 7.1 GB of memory but running seven in parallel on 1g.5gb instances uses 48.7 GB of memory which may be not be feasible for every user.

Figure 9a shows the aggregate amount of memory allocated to the processes running the workloads over time when training resnet_large. We see that the allocated memory increases over time for all of the experiments. Specifically, at each epoch start, between one and two additional gigabytes of memory are allocated per model running. We saw similar memory allocation characteristics for resnet_small and resnet_medium.

4.3.2 CPU utilization. The CPU-utilization was measured on the process level. Fig. 9b we see the average aggregate CPU utilization in percentages across all of our experiments, calculated using the procedure explained in section 3.2.3.

The main activities requiring CPU resources are reading training data from disk (in the case of resnet_medium and resnet_large), preprocessing the training data, transferring the training data to the GPU and keeping track of gradient information. From the chart we see that for the 7g.40gb one experiment, which represents the least constrained GPU computing environment, resnet_large requires significantly more CPU resources than resnet_small and resnet_medium. This is likely because resnet_large uses much larger images to train on which in turn also demand more preprocessing time. For the non-parallel experiments, we see that smaller GPU instances result in lower CPU utilization, e.g. resnet_large uses 198% CPU in 7g.40gb one whereas it only uses 119% CPU in 2g.10gb one. This makes sense since the smaller instances take longer to process each batch, which means that fewer images need to be read from disk and preprocessed per second.

For a parallel experiment with \( n \) concurrent workloads, we see that it uses approximately \( n \) times as much CPU processing power as the non-parallel version. For resnet_medium and resnet_large this relationship is almost exact, e.g. resnet_medium uses on average 85% CPU in 2g.10gb one and 257% CPU in 2g.10gb parallel which is almost exactly 3\( \times \)85%. For the resnet_small experiments, the relationship between the average CPU utilization of the non-parallel and parallel workloads is not as perfect as for the larger workloads. Instead, the parallel workloads utilize the CPU less than \( n \) times as much as for the non-parallel workloads. We are unsure what causes this.

An interesting thing to note is that in order to efficiently train seven concurrent machine learning models in parallel on 1g.5gb instances, a significant amount of CPU processing power is also required (in our case 630% with a very powerful CPU). This means that in addition to the A100 GPU and a substantial amount of RAM (see section 4.3.1) the system must also have fairly powerful CPU in order to get all the benefits of MIG.

4.4 Accuracy

We recorded the training and validation accuracies after each epoch for all of our experiments. Figure 10 provides an overview of the achieved accuracies. The charts show training and validation accuracy over time during a training run in a 7g.40gb instance and a smaller instance. In fig. 10a, the smaller instance is a 1g.5gb instance and in figs. 10b and 10c it is a 2g.10gb instance. It can be seen that resnet_small reaches a validation accuracy plateau of approximately 0.76 after about 1/5 of its total training time which is about five epochs. The resnet_medium and resnet_large models do not seem to reach their maximum validation accuracy within the five epochs that they were trained but their highest validation accuracies that they achieve were about 0.50 and 0.56 respectively. The figure shows that the size of the instance only impacts the total training time and not the achieved accuracy.
5 DISCUSSION

We begin this section by looking at some general trends across the results presented in Section 4. Then, we discuss the results from the replication of our experiments in addition to the some of the data used in Section 4 from them. Finally, we detail the challenges we experienced with DCGM tool covering both the 4g. 20gb profile case and the differences that we observed in some of the data.

5.1 General trends across results

Overall, the results demonstrate that parallel workload executions do not cause interference across workloads and, depending on the use case and needs, a smaller portion of the device can be a more efficient choice. Furthermore, instances with fewer allocated resources always report higher values for the hardware metrics than those with more resources. For example, in the case of hyperparameter optimization, a 1g. 5gb parallel instances would be able to both execute more workloads (in this with 3x throughput compared to the full device) and perform with higher device utilization. However, we found that using the smallest instance is not always possible. This was the case in our medium and large workloads. For such cases, we consider the 2g, 10gb and 3g, 20g as alternatives. On the other hand, we do not observe any throughput increase associated with the parallel runs over the isolated run for the medium and large cases.

Throughout the small workload experiments, the differences across the different MIG instance performance metrics are more expressed than the workloads operating at larger scale, following our expectations. When a workload dramatically grows in scale, the availability of more resources becomes beneficial. In our case, we use the same batch sizes. However, a bigger model requires more computation per batch iteration or data item. This can be expressed in the increased computation per unit time, which increases the utilization of the GPU. We suggest that this higher utilization reduces the variation in performance across instances. For that reason, we observed the small workload executions in more detail than the medium and large ones. Furthermore, we recognize the implications of increased input size to completion resulting from profiles with lower resources. A consideration in that area was presented separately in analysis of time to completion (see Section 4.1).

5.2 Results of replicated experimental runs

As mentioned in Section 3.4, we replicated each experimental even though results reported in Section 4 are from a single run. These replicated runs of experiments shows very similar or nearly identical results to the initial ones.

5.3 Challenges in metric collection

Even though the 4g, 20gb profile is a valid profile, we were limited in the information we could collect for this metric using DCGM tool. Some metrics could not be properly read with this profile. Furthermore, for some other runs, the last few seconds of a workload execution reported zero values for GRACK, SMACK, and SMOCC. The DRAMA metric showed an anomaly where it would present zero or near-zero values for the last few seconds of each run in some other cases. Since we did not further investigate the reasons for zero values toward workload completion, we considered the median values to be a more accurate representation of these metrics and helping us dealing with these reporting challenges.

6 CONCLUSION

In this paper, we performed a performance characterization for a modern GPU device that has hardware support, called Multi-Instance GPU (MIG), to split the GPU into multiple logical instances. Our results demonstrate that MIG is most useful for smaller workloads that cannot fully saturate the whole GPU. Executing the small workload on the smallest GPU instance, 1g. 5gb, resulted in significant increases in GRACK, SMACK and SMOCC. Although training is overall slower, more work can be done per unit of time by executing workloads in parallel on multiple GPU instances. We find no performance impact associated with co-location of workloads in separate GPU instances. Across all of our instance-level metrics, we see no difference between running one workload at a time and running multiple workloads in parallel. This highlights that MIG, even though still maturing, is a promising technology for workload co-location on GPUs.

In this work, we scoped our analysis on homogeneous instances and workloads when testing MIG. As future work, an investigation of more asymmetrical / heterogeneous instances and workloads would be important. In addition, we limited our focus on training using one GPU, since MIG doesn’t allow distributed training. Observing MIG while running other workloads on other GPUs on the same device may also be promising, as in a data center setting, many workloads can be co-located not only on the same GPU but also on the same server. Furthermore, the MIG tool-chain is still young and maturing. One should keep their tool chain up-to-date as much as possible for doing performance characterization studies such as this one to get the latest functionality.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In OSDI. 165–283.
[2] Robert Adolf, Sakeeth Rama, Brandon Reagen, Gu-Youe Wei, and David Brooks. 2016. Fatshark: reference workloads for modern deep learning methods. In ISWC. 1–10.
[3] AMD. 2022. AMD EPYC 7742. https://www.amd.com/en/products/cpu/amd-epyc-7742.
[4] Baidu Research. 2016. DeepBench. https://github.com/baidu-research/DeepBench.
[5] Sebastian Baunsgaard, Sebastian Benjamin Wrede, and Pinar Toprak. 2020. Training for Speech Recognition on Coprocessors. In ADMS. 1–6.
[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In Advances in Neural Information Processing Systems, Vol. 33. 1877–1901.
[7] Tianshi Chen, Yunji Chen, Marcus Duranton, Qi Guo, Atif Hashmi, Mikko Lipasti, Andrew Nere, Shi Qiu, Michele Sebag, and Olivier Temam. 2012. BenchNN: On training for Speech Recognition on Coprocessors. In ADMS. 1–6.
[8] Patrik Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A Downsampling Variant of ImageNet as an Alternative to the CIFAR datasets. CoRR arXiv (2017).
[9] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In ASPLOS. 127–144.
Deep Learning Training on Multi-Instance GPUs

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In CVPR. 770–778.

[11] Benjamin Hindman, Andy Korwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In NSDI. 295–308.

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In CVPR. 2261–2269.

[13] Alexandros Kolonias, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter Pietzuch. 2019. Crossbow: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers. PVLDB 12, 11 (2019), 1399–1412.

[14] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. Technical Report. University of Toronto.

[15] NVIDIA. 2021. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/.

[16] Shervin Minaee, Yuri Y. Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2021. Image Segmentation Using Deep Learning: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence 44, 7 (2021), 3523–3542.

[17] Konstantinos Nikas, Nikola Papadopoulos, Dimitra Giatrissi, Vasilios Karakostas, Georgios Goumas, and Nectarios Koziris. 2019. DICER: Diligent Cache Partitioning for Efficient Workload Consolidation. In ICFP.

[18] NVIDIA. 2015. Achieved Occupancy. Technical Report. NVIDIA. https://docs.nvidia.com/gameworks/content/developer-tools/desktop/analysis/report/cudaexperiments/kernelevel/achievedoccupancy.htm.

[19] NVIDIA. 2021. NVIDIA Multi-Instance GPU User Guide. NVIDIA. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/.

[20] NVIDIA. 2021. NVIDIA Multi-Instance GPU User Guide Documentation. Technical Report. NVIDIA. https://docs.nvidia.com/datacenter/dgem/latestd/gcm-user-guide/.

[21] NVIDIA. 2022. Data Center GPU Manager Documentation. Technical Report. NVIDIA. https://docs.nvidia.com/datacenter/dgemlatest/dcm-user-guide/.

[22] NVIDIA. 2022. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/.

[23] Kevin Pouget. 2021. Using NVIDIA A100’s Multi-Instance GPU to Run Multiple Workloads in Parallel on a Single GPU. https://cloud.redhat.com/blog/using-nvidia-a100s-multi-instance-gpu-to-run-multiple-workloads-in-parallel-on-a-single-gpu.

[24] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattsson, Guenther Schmale, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Igdunji, Thomas B. Jablin, Jeff Jao, Tom St John, Pankaj Kanwar, David Lee, Jeffrey Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gernady Pekhimenko, Arun Tejasve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In ISCA. 446–459.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. PJC 115, 3 (2015), 211–252.

[26] Lars Schmarje, Monty Santarossa, Simon-Martin Schroder, and Reinhard Koch. 2021. A Survey on Semi-, Self- and Unsupervised Learning for Image Classification. IEEE Access 9 (2021), 82146–82168.

[27] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 7587 (2016), 484–489.

[28] Cheng Tan, Zhihao Li, Jian Zhang, Yu Cao, Sikai Qi, Zherui Liu, Yibo Zhu, and Chuanxiong Guo. 2021. Serving DNN Models with Multi-Instance GPUs: A Case of the Reconfigurable Machine Scheduling Problem. CoRR arXiv (2021).

[29] Jin-Hua Tao, Zi-Dong Du, Qi Guo, Hui-Ying Lan, Lei Zhang, Sheng-Yuan Zhou, Cong Liu, Hai-Feng Liu, Shan Tang, Allen Rush, William Chen, Shao-Li Liu, Yun-Ji Chen, and Tian-Shi Chen. 2018. BenchLP: Benchmarking Intelligence Processors. Journal of Computer Science and Technology 33 (10 2018), 1–23.

[30] Tensorflow 2022. Use a GPU. Tensorflow. https://www.tensorflow.org/guide/gpu, Accessed: 2022-05-29.

[31] Transaction Processing Performance Council (TPC) 2021. TPC Express AI (TPCx- AI). https://www.tpc.org/tpcx-ai/default5.asp.

[32] Shang Wang, Peiming Yang, Xuyuan Zheng, Xin Li, and Gernady Pekhimenko. 2021. Horizontally Fused Training Array: An Effective Hardware Utilization Squeezer for Training Novel Deep Learning Models. In MLSys. 1–15.

[33] Hongyu Zhu, Mohamed Akroot, Bojian Zheng, Andrew Pelegris, Anand Jayarajan, Amir Phanishayee, Bianca Schroeder, and Gernady Pekhimenko. 2018. Benchmarking and Analyzing Deep Neural Network Training. In ISWC. 88–100.