Performance Characteristics of Virtualized GPUs for Deep Learning

Scott Michael*†, Scott Teige**, Junjie Li‡, John Michael Lowe§, George Turner¶ and Robert Henschel‖
Research Technologies
Indiana University
Bloomington, IN, USA
Email: *scamicha@iu.edu, **steige@iu.edu, ‡lijunj@iu.edu, §jomlowe@iu.edu
¶turnerg@iu.edu, ‖henschel@iu.edu, †Corresponding Author

***Abstract*—As deep learning techniques and algorithms become more and more common in scientific workflows, HPC centers are grappling with how best to provide GPU resources and support deep learning workloads. One novel method of deployment is to virtualize GPU resources allowing for multiple VM instances to have logically distinct virtual GPUs (vGPUs) on a shared physical GPU. However, there are many operational and performance implications to consider before deploying a vGPU service in an HPC center. In this paper, we investigate the performance characteristics of vGPUs for both traditional HPC workloads and for deep learning training and inference workloads. Using NVIDIA's vDWS virtualization software, we perform a series of HPC and deep learning benchmarks on both non-virtualized (bare metal) and vGPUs of various sizes and configurations. We report on several of the challenges we discovered in deploying and operating a variety of virtualized instance sizes and configurations. We find that the overhead of virtualization on HPC workloads is generally $< 10\%$, and can vary considerably for deep learning, depending on the task.**

***Index Terms*—Deep Learning, High Performance Computing, Virtualization**

I. Introduction

Approaches using deep learning frameworks are becoming more and more commonplace in a variety of scientific workflows. Aspects of deep learning have been integrated into scientific workflows in genomics [1], precision health [2], climate science [3], astrophysics [4], material science [5], chemistry [6], and many other fields. As these types of workflows become more prevalent in high performance computing (HPC) centers, the need for flexibility in the deployment and allocation of resources is becoming more pronounced. There are currently several strategies for using multiple graphics processing units (GPUs) within a single node [7], [8], and across many nodes [9], [10], to do large-scale distributed training of very large models with novel data sets.

Less attention has been given to strategies to optimize the utilization of the the GPU deployments that are typically seen in HPC centers today. While there are many deep learning training workflows that can take advantage of multiple state-of-the-art GPU cards (currently NVIDIA V100) in a node, there are also many that cannot make full use of even a single card. Additionally, inference workflows seldom require the resources of a full GPU. While there are some hardware offerings specifically geared toward inference, these are not always available in HPC centers.

This paper explores the possibility of using virtualized GPUs (vGPUs) to address the various workflows that do not currently make optimal use of GPU resources that are available in HPC centers today. We conduct a performance study looking at the performance characteristics of vGPUs as compared to non-virtualized GPUs (i.e., "bare metal"). In section II, we discuss a few potential workflows that vGPUs are well suited to, and ways that vGPUs can be used to "right-size" the resource for the workflows being run. In section III, we describe the methodology for the performance analysis, including the details for the hardware, virtualization layers, and software stack used to run the benchmarks. We also describe the benchmarks used and key settings and configurations for the experimental setup. Section IV provides the detailed results from the benchmarking and performance analysis, and sections V and VI wrap up with discussions of the results and future work to be undertaken.

II. Motivation

As deep learning workflows become more and more common in a variety of scientific and research domains, HPC centers from the individual lab level of the principal investigator to the national lab level are grappling with how best to provide GPU resources to researchers for these workflows. There are several ways in which deep learning workflows differ from a more traditional HPC workload based on batch scheduled modelling and simulation.

Unlike many simulation workloads, the capability to perform a checkpoint and restart in a deep learning workflow is highly dependent on how the workflow is designed. Although it is possible to include a checkpointing step in the training phase of a workflow, many researchers do not include this step because they plan to run on a system that they have exclusive use of for a long period of time. Another difference is that in the training process, many researchers prefer to tune their model hyperparameters in an interactive fashion. In many cases, researchers want to examine the models they have constructed using a graphical interface (e.g., tensorboard). There is also an inherent dichotomy in most deep learning workflows between training and inference. Although there are some studies that require only training to find model parameters of interest and others that use pre-trained models, in most cases, a workflow will use both training and inference. However, the hardware resources required for a typical training run are generally much greater than those needed for inference with a similarly sized model. This leads to the challenge of providing hardware resources that have enough computational power to train models in a reasonable time, but that can be allocated for inference and efficiently use the resource. Finally, the basic frameworks supporting most deep learning workflows, both training and inference, are not designed to scale to multiple GPUs within a node or multiple nodes without additional

978-1-6654-2272-7/20/$31.00 ©2020 IEEE
DOI 10.1109/SuperCompCloud51944.2020.00008

framework libraries or custom code. The parallelization strategies that one would employ for training and inference are inherently different with inference being an embarrassingly parallel process, while training requires communication among computed elements for evaluation and iteration of model parameters. All of these differences from a traditional HPC simulation workload combine to make it difficult for a batch scheduled HPC machine to meet all of the needs of a researcher using deep learning algorithms.

One way to address several of these challenges is to provide virtualized GPUs on virtual machines (VMs) via a cloud based infrastructure. The VMs and vGPU allocations can then be appropriately sized to facilitate the workload in question. In addition, the VM can be accessed in an interactive fashion and a researcher can work with the resource in much the same way that they would a dedicated interactive resource. This approach gives both the end user and the resource provider a good deal of flexibility in terms of resource allocation and delivery. The goal of this study is to investigate the feasibility of deployment, discover any potential problems in operations, and measure the potential performance impact of using vGPUs in a production cloud environment. While the performance overhead due to virtualization has been measured in the past [11], and continues to improve over time, there is very little data as to the performance impact of virtualization on GPUs. We elucidate several of the issues we uncovered, and find that overall the overhead of virtualization on HPC workloads is generally $< 10\%$, and can vary considerably for deep learning, depending on the task.

III. Methodology

We used the National Science Foundation (NSF) Jetstream resource [12], [13] to evaluate vGPUs in a production environment. Jetstream is a NSF-funded science and engineering cloud resource that has been widely used by the NSF community over the past several years. The production side of Jetstream is composed of CPU only nodes that host VMs for researchers. Although many HPC resources support container based workflows via Singularity or a similar container technology, the use of VMs in HPC is less widespread. In the XSEDE ecosystem Jetstream is the only allocatable resource that supports VMs [14]. The potential performance implications for containers as opposed to VMs managed via a hypervisor are somewhat different [15]. Indiana University has augmented Jetstream with six nodes containing GPUs as a testbed for evaluating the performance and operations of virtualized GPUs. We obtained and provisioned several nodes in Jetstream, each with dual CPUs and four NVIDIA V100 GPUs. These nodes were configured in several different ways including bare metal, one vGPU per physical GPU (four vGPUs per node), two vGPUs per physical GPU (eight vGPUs per node), and four vGPUs per physical GPU (16 vGPUs per node). We then ran a variety of benchmarks on each of these instances in both ”best effort” and ”fixed” modes (see section III-B for definitions of these modes). It should be noted that at this time the virtualization software used for these tests required that the GPUs be partitioned in the same way across an entire node. It does not allow, for instance, an administrator to partition one physical GPU into two vGPUs and another physical GPU into four vGPUs if those physical GPUs are in the same node. In the cases where there was more than one vGPU per physical GPU we ran both a single instance of the benchmark and multiple instances with one per vGPU. Although we only present results for a particular hardware configuration, the hardware we have detailed is sufficiently general purpose that we believe that our results are broadly applicable. In this section, we detail the experimental setup for these benchmarks, including the hardware configuration, the virtualization configuration for each of the nodes with vGPUs, the software stacks used, and the benchmarks run.

A. Hardware specifications

The hardware used for the performance characterization included several Dell PowerEdge C4140 servers, each with dual Skylake Xeon Gold 6130 [16] CPUs and four NVIDIA V100-SXM2 16GB [17] GPUs. Nodes contain 192GB of memory from twelve 16GB DDR4-2666 RDIMMs. NVlink is present but not utilized, as it only provides GPU to GPU connections and the testing involved single GPUs. All nodes are connected with bonded dual 10Gbps ethernet in a fat-tree topology in a 2:1 oversubscription configuration. Testing inside and outside of virtual machines was backed by network based storage from a Ceph filesystem exported over NFS via a Ganesha user space NFS server. Both HPL and SPEC Accel were run on a Google Compute Platform (GCP) instance (`n1-standard-8`) with 8 vCPUs, 30GB of memory, and a single non-virtualized NVIDIA V100-SMX2 16GB.

B. Virtualization software

The GPUs are virtualized using NVIDIA's Virtual Data Center Workstation [18] (vDWS) software, which is available for NVIDIA V100 GPUs. The only HPC data center GPUs currently supported by the vDWS software are the P100 and the V100. We chose to conduct experiments with the V100 cards as they are the most recently released cards by NVIDIA. The vDWS software is based upon the Linux kernel's Virtual Function I/O [19] (VFIO). VFIO is a Linux kernel framework which allows for safe, non-privileged userspace drivers. Utilizing isolation features of modern input–output memory management units (IOMMUs), VFIO kernel drivers create an allowed range of DMA, I/O access, and interrupts that can be assigned to a virtual machine hypervisor running as a user space process. This is a logical extension of using VFIO to implement a user space driver. These virtualized device functions are then passed through by the hypervisor to guest VM kernel drivers. A two-part driver approach requires matching drivers inside the guest VM and in the underlying host. The performance impact of using VFIO is minimized by the hardware implementation of IOMMU isolation features, lightweight drivers implementing VFIO, and efficient passthrough of these virtualized functions by the hypervisor. Use of a unified memory programming model is not allowed in the current software release. Guests are allotted some fixed portion of the GPUs frame buffer but are able to use all CUDA cores on a time division multiplexing basis. Scheduling policies allow an administrator to choose between allocating all available time slices to vGPUs with work queued (best effort), allocating an equal number of slices to to each vGPU assigned to a GPU (equal share), or allocating slices proportional to the fraction of the frame buffer assigned to the vGPU (fixed). For example, if a 16GB GPU was assigned 3 vGPUs with 4GB each and only one vGPU was actively being used, best effort gives all slices to the active vGPU, equal share would give the active vGPU 1/3 of the time slices, and fixed would give 1/4 of the time slices to the active vGPU. For most

of these tests, the scheduler policy was set to fixed. The vDWS framework allows for a single V100-SMX2 16GB card to be divided into 1, 2, 4, 8, or 16 vGPUs, however the smallest vGPU we tested was 1/4 of the card due to memory constraints.

C. Application software and benchmarks

We ran several different benchmark suites on the bare metal GPUs and vGPUs in VMs. Two of the benchmarks (HPL and SPECAccel) were designed to determine baseline performance of the bare metal GPUs as compared to that of the vGPUs. The other benchmarks were targeted at deep-learning-specific workloads, and included two components of MLperf v0.5[1] and an inference task developed at Indiana University.

High Performance Linpack (HPL) [20] is widely used to determine the performance of a computational platform on solving dense linear systems. In our case, it is used to measure the baseline performance of the vGPUs as compared to the bare metal systems. The HPL binary was compiled using the Intel compiler, version 10.0 of the CUDA libraries, and OpenMPI v3.1.

The SPEC Accel suites[21] test performance with computationally intensive parallel applications running under the OpenCL, OpenACC, and OpenMP 4 target offloading APIs. The suite exercises the performance of the accelerator, host CPU, memory transfer between host and accelerator, support libraries and drivers, and compilers. For these experiments we used version 1.2 of the SPEC Accel OpenACC suite to gauge performance. This suite contains 15 different real-world scientific applications or proxy apps written in Fortran and C covering multiple science domains. These applications are run and scored independently, and an aggregate score is produced for the suite. This score is in comparison to a reference machine; see the SPEC Accel website [22] for more information. The SPEC Accel suite was compiled with the PGI compiler v19.4 and version 10.1 of the CUDA libraries.

One of the more commonly used deep learning benchmarks today is the MLPerf benchmark. MLPerf borrows several ideas from SPEC high performance benchmarks, but has a different set of rules for submission and execution of benchmarks. It also focuses solely on deep learning workloads. Submissions are allowed to use any deep learning framework and any hardware configuration they want to submit scores; however, for the closed division, they have to use the models specified for each of the tasks. In this study we used the image classification and object detection, heavy-weight tasks of MLPerf Training v0.5. For image classification we used TensorFlow v1.14.0 with CUDA v10.1 and the cuDNN v7 library, and for object detection we used a container specified in the MLPerf documentation. Although there is a MLPerf Inference v0.5 benchmark, we found this benchmark even more challenging to deploy and execute than the MLPerf Training v0.5 benchmark. We used the Oxford Visual Geometry Group (VGG) 16-layer pretrained model for image classification (VGG16) [23] to classify a subset of ImageNet 2013 images [24].

IV. Results

In this section, we provide results from the various benchmarks. We also detail several issues we encountered when performing a few of the benchmarks that required workarounds to obtain the results presented.

[1]No results presented in this work have been verified by MLPerf. MLPerf name and logo are trademarks. See www.mlperf.org for more information.

TABLE I: HPL scores

Instance	Scheduling Policy	HPL Score (TFlops)
No virtualization	–	4.71
16Q (full GPU)	best effort	4.31
8Q (half GPU)	fixed	2.66
Google Cloud V100	–	5.63

TABLE II: SPEC Accel scores

Instance	Scheduling Policy	SPEC Accel Score
No virtualization	–	13.3
16Q (full GPU)	best effort	13.2
8Q (half GPU)	best effort	13.2
8Q (half GPU)	fixed	7.09
4Q (quarter GPU)	fixed	3.99
Google Cloud V100	–	11.3

The HPL benchmark was run on bare metal, 16Q, 8Q, and in a GCP instance (see tables I and II for definitions). We were unable to obtain results for the 4Q instance. The results for the HPL runs are presented in Table I. The V100-SMX2 cards have a peak double precision floating-point performance of 7.8 TFLOP/s. Initially, HPL provided much poorer results than what is presented in Table I. This was due to the fact that turbo had been turned on for only the card that the HPL test was running on, which did not result in expected behavior even though `nvidia-smi` reported that the card was operating at turbo frequency. When turbo was enabled for all four cards in the node, we were able to achieve 4.71 TFLOP/s (60% of peak) on bare metal. The 16Q instance run in best effort mode gave 8% less performance than bare metal. The 8Q instance gave nearly the same performance as 16Q when run in best effort, however when run in fixed mode, constraining the run to using only half of the card's resources, the performance dropped to 62% of the 16Q instance, or 54% of the bare metal numbers. We were able to achieve 5.63 TFLOP/s on the GCP instance for HPL.

The SPEC Accel suite was run on bare metal, 16Q, 8Q, 4Q, and a GCP instance. Results for the SPEC Accel benchmark runs are presented in Table II. The best currently-published result using a 16GB V100 card is 13.2. One item of note for the SPEC Accel benchmarks is that the suite requires a fairly small amount (2GB) of device memory. On bare metal we obtained 13.3, which is slightly better than the best published result. This is likely due to the fact that the 13.2 result was obtained with a V100-PCIE 16GB which has a slightly lower peak performance than the V100-SMX2. The 16Q instance was nearly equivalent to the bare metal run ($<$ 1% difference) and the best effort 8Q score was identical to the 16Q score. When run in fixed mode, the 8Q instance scored 54% of the 8Q best effort and 16Q instances. The 4Q fixed instance scored 30% of the 8Q best effort and 16Q instances. All of the 8Q and 4Q runs were performed with only a single instance running on each card. The SPEC Accel suite was also run on the GCP instance and scored 11.3, which is 14% less than the 16Q virtualized instance.

Two components of the MLPerf Training v0.5 benchmark suite were run: image classification and object detection, heavy-weight. The image classification task was run with TensorFlow v1.14.0 that was built using Bazel 0.25.1 and Python 3.7.3. The object detection task was run using a docker container specified in

TABLE III: MLperf 0.5 Image Classification Training Run Times

Instance	Scheduling Policy	Run time (minutes)
No virtualization	–	3964
16Q (full GPU)	fixed	4231
No virtualization (4xV100)	–	1294

the MLPerf object detection repository. This container uses pyTorch v1.0.1, CUDA v10.0, and cuDNN v7. Both software installations are on node local storage. The ImageNet data set for image classification and the COCO data set for object detection were stored on the Ceph filesystem exported over NFS via Ganesha from section III-A. Tables III and IV report the run times for these components on bare metal and 16Q instances.

The reference machine for MLPerf is "16 CPUs, one Nvidia P100"[2] with a run time on image classification of 8831 minutes. There are currently no results published on on the MLPerf website for a single V100. The P100 has a peak single precision floating-point performance of 9.3 TFLOP/s for the PCIE based card (the MLPerf documentation does not specify PCIE or NVLINK/SMX2) and the V100 has a peak single precision floating-point performance of 15.7 TFLOP/s for the SMX2 card. The timing differential for our V100 on bare metal as compared to the P100 reference machine (2.75x) is better than the P100/V100 single precision performance differential (1.69x). This is a bit surprising considering the MLPerf reference machine is a cloud based instance with "local" storage and our experiment uses NFS mounted Ceph storage.

Running the image classification training task on the 16Q fixed instance was 7% slower than the bare metal run, roughly in line with the HPL numbers. In addition, we performed a run of image classification on bare metal using all four of the V100s. When compared to the single card bare metal run, using all four GPUs in a node for the image classification training task was 3.06 times faster. While the image classification benchmark ran without much issue on bare metal, we did encounter problems with the run on 16Q. Investigation of the vGPU via `nvidia-smi` revealed that the device was presenting more memory than was physically available. This was the likely culprit for several failed runs, which probably tried to use device memory that was not actually present. After modifying the code as in Listing 1 we were able to run the benchmark limited to 95% of the memory reported by `nvidia-smi`.

Listing 1: Limiting memory usage in TensorFlow

```
mem_frac = 0.95
gpu_options = tf.GPUOptions
    (per_process_gpu_memory_fraction
    =mem_frac)
sess = tf.Session(config=tf
    .ConfigProto(gpu_options=gpu_options))
```

The MLPerf object detection, heavy-weight training task was also evaluated. The reference machine run time for object detection is 5000 minutes. Unlike the image classification training task, the timing differential between the P100 and V100 cards for object detection (1.56x) is slightly less than the single precision performance differential (1.69x). This may be due to the fact that we used containerized runtimes that more closely matched the reference machine run. It is not clear exactly what version of TensorFlow was used for the reference machine run cited on the MLPerf website, but it is almost certainly an older version than the v1.14 that we used for image classification. The 16Q best effort instance ran slightly slower (6%) than the bare metal, which is a similar result to the image classification training task. Unlike the image classification training benchmark that was written in TensorFlow, the object detection training benchmark uses PyTorch. We were unable to find an easy way to limit the memory usage in PyTorch as in Listing 1 with TensorFlow. The object detection training task was able to run on the 16Q instance, but failed to run to completion on the 8Q instance, likely due to memory issues. We were able to run the task for several hours and extrapolate the completion time. This estimate is provided in Table IV and is 2.28 times longer than the 16Q run time.

Although a MLPerf inference benchmark is currently available, the benchmark is fairly challenging to run. Instead of using this, we evaluated an inference task of our own design. A pre-trained model, VGG-16 (for Visual Geometry Group, Oxford University, 16 layers deep), [23] was used for these tests along with the validation data from the 2013 ImageNet large scale visual recognition challenge [24]. The data set contained 20,121 images.

Table V gives a summary of the conditions and results of this measurement. Multiple runs of each configuration were performed. Run times are presented in seconds with the standard deviation for 5 runs. Because inference tasks tend to be less resource-intensive than training tasks we ran a series of tests to see how running multiple inference tasks in parallel affected performance. The rows with 2 tasks give the results (in seconds) when two classification tasks were run simultaneously on two copies of the data cached in shared memory. To run two tasks on a single bare metal or virtual GPU, the allocation of memory on the GPU was manually configured via modification of the Python script initiating the benchmark. A nominal 40% of the memory was allocated to each task. It was found that this allocation was insufficient to run two tasks on a single 8Q node. For the 8Q instances, 1 task represents a single task running on one 8Q node, while 2 tasks represent two 8Q nodes sharing the same physical GPU and each running a single task.

For inference, we see that the bare metal system performs better than the 16Q (6-10%), although the performance gap narrows a bit for two simultaneous tasks. Although there is some overhead in running two tasks simultaneously, which is likely due to PCIE contention, it is only 20-25%. We also found that the run times were incredibly consistent with a $< 1\%$ run to run variation. When only half of the GPU is allocated in the 8Q instance, we see that the run time for even a single task takes much longer than either the bare metal (57% longer) or the 16Q (43% longer). However, with two 8Q instances simultaneously running a single task on one physical GPU, we see slightly faster completion times than a single task, and only moderately longer run times than two tasks running on bare metal (20% longer) or 16Q (14% longer).

In all cases considered the non-virtualized configuration outperformed the virtualized system. Comparing columns two and three of table VI shows the difference to be 6-10% comparing a physical machine using 100% of a GPU to a

[2]MLPerf v0.5 Training Closed; Retrieved from www.mlperf.org 11 September 2019, entry 0.5-1. MLPerf name and logo are trademarks. See www.mlperf.org for more information.

TABLE IV: MLperf 0.5 Object Detection Training Run Times

Instance	Scheduling Policy	Run time (minutes)
No virtualization	–	3037
16Q (full GPU)	best effort	3206
8Q (half GPU)	fixed	6927[1]

[1] Extrapolated run time based on run that failed after several hours.

TABLE V: Image Classification with Pre-trained Model

Instance	Scheduling Policy	Tasks	Run time (seconds)
No virtualization	–	1	179.8 $\pm$ 1.2
No virtualization	–	2	225.4 $\pm$ 1.8
16Q (full GPU)	fixed	1	198.1 $\pm$ 1.2
16Q (full GPU)	fixed	2	239.5 $\pm$ 2.0
8Q (half GPU)	fixed	1	283.1 $\pm$ 0.9
8Q (half GPU)	fixed	2	272.4 $\pm$ 2.7

similarly configured virtual machine. Comparison of columns three and five show a substantial performance degradation (43%) when only half of the GPU is allocated. We speculate that this configuration provided sufficient resources to allow the application to run but not to run efficiently.

V. Discussion

The provisioning of vGPUs offers a great deal of flexibility for both HPC service providers and users. It allows users to select vGPU instances that are "right sized" for their particular workflows and service providers to provide an offering that is not overkill for lighter weight workflows such as inference based workflows. Virtualization may also allow machines with large numbers of GPUs to be used in a larger variety of ways than is currently possible with scheduling on bare metal. For example, a node with 8 GPUs can be presented as eight VMs with one physical GPU each, or as many as 64 VMs with 1/8 of a physical GPU each. However, there are many caveats to deploying such a system. First is that all of the GPUs in a node must be divided in the same way: a node with four physical GPUs cannot currently have one provisioned as a 16Q vGPU and another physical GPU provisioned as two 8Q vGPUs. To switch the node from one virtual configuration to another (e.g., 16Q to 8Q) requires a system reboot. One must also consider how to partition the rest of the resources on a node. For a system with eight physical GPUs and 32 vGPUs to have 128GB of main system memory per VM requires over 1TB of system memory on the node. Similar considerations must be made for vCPU cores as well.

TABLE VI: Image Classification with Pre-trained Model *units are seconds*

	Physical machine	16Q	2 8Q nodes	1 8Q node
2 tasks	225.4$\pm$1.8	239.5$\pm$2.0	272.4$\pm$2.7	not possible
1 task	179.8$\pm$1.2	198.1$\pm$1.3	N/A	283.1$\pm$0.9

A. HPC benchmarks

Overall, the virtualized GPUs performed fairly well on HPC tasks. When running virtualized on the full card, the performance overhead from virtualization was $< 10\%$, and in the case of SPEC Accel, the overhead was negligible. When the vGPU was limited to half or a quarter of the card's resources, the performance stayed above half or a quarter of the bare metal numbers. It should be noted that all of the HPC benchmarks we ran had fairly modest device memory requirements, so device memory management was not a major issue in the way that it was for the deep learning benchmarks. It was only due to the fact that SPEC Accel requires 2GB of device memory that we were able to run it on the 4Q instance; all of the other benchmarks we attempted on the 4Q instance failed due to insufficient device memory.

B. Deep learning benchmarks

The MLPerf and inference benchmarks we ran were on par with the overhead seen in HPL, but fared a bit worse than SPEC Accel, with virtualization introducing a 6-10% overhead for the deep learning benchmarks. For the data we were able to collect for 8Q instances, the performance using half of the cards' resources was less than half of the bare metal runs. The larger challenge beyond the performance overhead is the need for explict memory management on the device. The fact that the vGPU reported more device memory than was physically available to it meant that workflows would fail in unanticipated ways. What would normally result in a "device out of memory" error message instead resulted in the run halting with no error output. Although we were able to work around this for the image classification training workflow, it required some code changes. We would anticipate that most codes using a moderate amount of device memory would need to explicitly manage the memory and should not rely on accurate memory information from tools such as `nvidia-smi` or the CUDA toolkit when running in a virtualized mode. We expect that this issue can be remedied through some combination of updates to the vDWS software and updates to the NVIDIA drivers.

VI. Conclusions and Future Work

The vGPU technology represents a very promising option that HPC service providers could utilize to increase the flexibility of their service offerings. We have shown that the virtualization overhead for both HPC workloads and deep learning workloads is relatively small. The device memory requirements of the researchers' workflows is probably the most critical variable that service providers should consider when evaluating the use of vGPUs.

Explicit memory management at the application level is easily applied but difficult to enforce in a production environment. Additionally, the implementation used for this work is tied specifically to TensorFlow. An effort to make such capability easily implementable in other popular frameworks is a possible direction for future work. Hopefully, future improvements to the NVIDIA drivers and vDWS system should obviate the requirement for explicit memory management on vGPUs.

It should be noted, however, that vDWS is only one approach to providing flexible allocations of GPU resources within a cluster. There are other approaches to this problem and the hardware executing workflows is rapidly evolving. For example, the newly released A100 cards support a new Multi-instance GPU (MiG) feature which allows for partitioning of GPUs at both the bare metal and virtual level. The details of the implementation of MiG likely have implications on performance and deployment characteristics that are distinct from vDWS and this comparison is worth investigating. This paper shows that there is an approach to solving it with an acceptably small impact on the efficiency of using GPUs. However, it is left to a future paper to perform an exhaustive evaluation of all approaches on all platforms.

The long-running MLPerf benchmark, when applied to one and four GPUs on the bare metal system, achieved a factor of three in performance as measured by execution time. This is a rather rapid deviation from ideal scaling, implying that use of multiple GPUs should be approached with caution. Additionally, it points out a need for instrumentation to determine the location and nature of bottlenecks and performance limitations.

The nature and cause of the observed deviation from ideal scaling is currently unknown; application (and development) of instrumentation seems to be a profitable future direction.

In addition to instrumentation at the vGPU level and transport layer between vGPUs, there are a number of other potential bottlenecks to investigate. Most deep learning workflows are sensitive to the rate at which data can be transported into the card. In this study we had data stored in several different storage locations (local memory, local disk, networked file systems), but did not test these storage platforms head to head for each of the workflows. In addition, there are other storage options that would be interesting to compare (e.g., NVMe storage).

We did not explore the performance impact of fully loading the physical systems with virtual instances and running simultaneous benchmarks (e.g., four 16Q instances on a single node all running an MLPerf task, or eight 8Q instances). Although the virtualization isolates much of the system from other VMs running on the same physical hardware, there are still shared resources (e.g., network bandwidth to file systems), which may introduce additional bottlenecks when the physical systems are fully utilized. These tests should be carried out in the future. There is also the possibility in future releases of vDWS of providing vGPUs that consolidate multiple physical GPUs (i.e., presenting two or more GPUs as a single logical device). This capability will only further flexibility in what can be offered to end users, and, when it is made available, its performance impact should be evaluated.

The benchmarks used for this work were MLPerf and an implementation of an ImageNet LSVR challenge winner. MLPerf required several days to complete on a single GPU system; the VGG inference benchmark took a few minutes. Other benchmarks [25] exist and should be investigated. We observe here that there is a need for realistic benchmarks that run in an amount of time intermediate to what was measured here, and that can be used to benchmark systems from a fraction of a V100 card up to 8 or more of the next generation of GPU devices. Additionally, small memory footprint benchmarks for systems configured to use only a fraction of a modern GPU should be developed and distributed.

ACKNOWLEDGEMENTS

Jetstream is supported by the National Science Foundation (award number 1445604) and Indiana University. Many thanks to Harmony Jankowski for her valuable contributions to improving this manuscript.

REFERENCES

[1] J. Zou, M. Huss, A. Abid, P. Mohammadi, A. Torkamani, and A. Telenti, "A primer on deep learning in genomics," *Nature Genetics*, vol. 51, no. 1, pp. 12–18, 2019. [Online]. Available: https://doi.org/10.1038/s41588-018-0295-5

[2] A. L. Beam and I. S. Kohane, "Big Data and Machine Learning in Health Care," *JAMA*, vol. 319, no. 13, pp. 1317–1318, 04 2018. [Online]. Available: https://doi.org/10.1001/jama.2017.18391

[3] Y. Liu, E. Racah, Prabhat, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. F. Wehner, and W. D. Collins, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *CoRR*, vol. abs/1605.01156, 2016. [Online]. Available: http://arxiv.org/abs/1605.01156

[4] D. George and E. A. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," *Phys. Rev. D*, vol. 97, p. 044039, Feb 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.97.044039

[5] S. M. Azimi, D. Britz, M. Engstler, M. Fritz, and F. Mücklich, "Advanced steel microstructural classification by deep learning methods," *Scientific Reports*, vol. 8, no. 1, p. 2128, 2018. [Online]. Available: https://doi.org/10.1038/s41598-018-20037-5

[6] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, pp. 1263–1272. [Online]. Available: http://dl.acm.org/citation.cfm?id=3305381.3305512

[7] R. Xu, X. Tian, S. Chandrasekaran, and B. Chapman, "Multi-GPU support on single node using directive-based programming model," *Sci. Program.*, vol. 2015, pp. 3:3–3:3, Jan. 2016. [Online]. Available: https://doi.org/10.1155/2015/621730

[8] Y. Pan, Y. Wang, Y. Wu, C. Yang, and J. D. Owens, "Multi-GPU graph analytics," *CoRR*, vol. abs/1504.04804, 2015. [Online]. Available: http://arxiv.org/abs/1504.04804

[9] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Tbd: Benchmarking and analyzing deep neural network training," 2018, cite arxiv:1803.06905. [Online]. Available: http://arxiv.org/abs/1803.06905

[10] H. Ma, F. Mao, and G. W. Taylor, "Theano-mpi: A theano-based distributed training framework," in *Euro-Par 2016: Parallel Processing Workshops*, F. Desprez, P.-F. Dutot, C. Kaklamanis, L. Marchal, K. Molitorisz, L. Ricci, V. Scarano, M. A. Vega-Rodríguez, A. L. Varbanescu, S. Hunold, S. L. Scott, S. Lankes, and J. Weidendorfer, Eds. Cham: Springer International Publishing, 2017, pp. 800–813.

[11] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 9–16.

[12] C. A. Stewart, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner, M. Vaughn, and N. I. Gaffney, "Jetstream: A self-provisioned, scalable science and engineering cloud environment," in *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, ser. XSEDE '15. New York, NY, USA: ACM, 2015, pp. 29:1–29:8. [Online]. Available: http://doi.acm.org/10.1145/2792745.2792774

[13] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "Xsede: Accelerating scientific discovery," *Computing in Science Engineering*, vol. 16, no. 5, pp. 62–74, Sep. 2014.

[14] Xsede resource information. [Online]. Available: https://portal.xsede.org/allocations/resource-info

[15] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *2015 IEEE International Conference on Cloud Engineering*, March 2015, pp. 386–393.

[16] *6th Generation Intel Core Processor Family: Datasheet-Volume 1*, Intel, 8 2015. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents\/datasheets/desktop-6th-gen-core-family-datasheet-vol-1.pdf

[17] *NVIDIA TESLA V100 GPU ARCHITECTURE*, NVIDIA, 8 2017. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[18] *NVIDIA QUADRO Virtual Data Center Workstation*, NVIDIA, 4 2019. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/documents/185532_NVIDIA_Quadro%20vDWS_SolutionOverview_NV_US_WEB.pdf

[19] Linux Foundation. VFIO - "Virtual Function I/O". [Online]. Available: https://www.kernel.org/doc/Documentation/vfio.txt

[20] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - a portable implementation of the high-performance Linpack benchmark for distributed-memory computers. [Online]. Available: http://www.netlib.org/benchmark/hpl/

[21] G. Juckeland, W. Brantley, S. Chandrasekaran, B. Chapman, S. Che, M. Colgrove, H. Feng, A. Grund, R. Henschel, W. Hwu, H. Li, M. Müller, W. Nagel, M. Perminov, P. Shelepugin, K. Skadron, J. Stratton, A. Titov, K. Wang, M. Van Waveren, B. Whitney, S. Wienke, R. Xu, and K. Kumaran, "SPEC ACCEL: A standard application suite for measuring hardware accelerator performance," in *High Performance Computing Systems*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in

Artificial Intelligence and Lecture Notes in Bioinformatics), S. Hammond, S. Jarvis, and S. Wright, Eds. Springer-Verlag, 1 2015, pp. 46–67.
[22] Standard Performance Evaluation Corporation. SPEC ACCEL. [Online]. Available: https://www.spec.org/accel/
[23] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556
[24] O. Russakovsky[1], J. Deng[1], H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and Li Fei−Fei, [1](equal contribution), "Imagenet large scale visual recognition challenge," 2014. [Online]. Available: https://arxiv.org/abs/1409.0575
[25] C. Bourrasset, F. Boillod-Cerneux, L. Sauge, M. Deldossi, F. Weillenreiter, R. Bordawekar, S. Malaika, J.-A. Broyelle, M. West, and B. Belgodere, "Requirements for an enterprise AI benchmark," in *Tenth TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC 2018)*, 2018. [Online]. Available: http://cognitive-science.info/wp-content/uploads/2018/08/Requirements_for_a_Enterprise_AI_Benchmark.TPCTC_.20180831c.pdf