Benchmarking the Performance of Accelerators on National Cyberinfrastructure Resources for Artificial Intelligence / Machine Learning Workloads

Abhinand S. Nasari
HPRC, Texas A&M University, College Station TX
abhinand@tamu.edu

Hieu T. Le
Department of Electrical and Computer Engineering, Texas A&M University, College Station TX
hieult@tamu.edu

Richard Lawrence
HPRC, Texas A&M University, College Station TX, ,
rlawrence@tamu.edu

Zhenhua He
HPRC, Texas A&M University, College Station TX
happidence1@tamu.edu

Xin Yang
HPRC, Texas A&M University, College Station TX
karen890@tamu.edu

Mario M. Krell
Graphcore Inc., Palo Alto CA
mmk@graphcore.ai

Alex Tsyplikhin
Graphcore Inc., Palo Alto CA
alext@graphcore.ai

Mahidhar Tatineni
San Diego SuperComputing Center, University of San Diego, San Diego, CA
mahidhar@sdsc.edu

Tim Cockerill
Texas Advanced Computing Center, University of Texas at Austin, Austin, TX
cockerill@tacc.utexas.edu

Lisa M. Perez
HPRC, Texas A&M University, College Station TX
perez@tamu.edu

Dhruva K. Chakravorty
HPRC, Texas A&M University, College Station TX
chakravorty@tamu.edu

Honggao Liu
HPRC, Texas A&M University, College Station TX
honggao@tamu.edu

ABSTRACT

Upcoming regional and National Science Foundation (NSF)-funded Cyberinfrastructure (CI) resources will give researchers opportunities to run their artificial intelligence / machine learning (AI/ML) workflows on accelerators. To effectively leverage this burgeoning CI-rich landscape, researchers need extensive benchmark data to maximize performance gains and map their workflows to appropriate architectures. This data will further assist CI administrators, NSF program officers, and CI allocation-reviewers make informed determinations on CI-resource allocations. Here, we compare the performance of two very different architectures: the commonly used Graphical Processing Units (GPUs) and the new generation of Intelligence Processing Units (IPUs), by running training benchmarks of common AI/ML models. We leverage the maturity of software stacks, and the ease of migration among these platforms to learn that performance and scaling are similar for both architectures. Exploring training parameters, such as batch size, however finds that owing to memory processing structures, IPUs run efficiently with smaller batch sizes, while GPUs benefit from large batch sizes to extract sufficient parallelism in neural network training and inference. This comes with different advantages and disadvantages as discussed in this paper.As such considerations of inference latency, inherent parallelism and model accuracy will play a role in researcher selection of these architectures. The impact of these choices on a representative image compression model system is discussed.

CCS CONCEPTS

• **Machine Learning**; • **Hardware Acceleration**; • **Benchmarking**;

KEYWORDS

ResNet50, ACES (Accelerating Computing for Emerging Sciences), Expanse, Graphics Processing Unit, Intelligence Processing Unit, PopVision, Classification, Convolution Neural Network, Optimization, Frontera, LoneStar6

ACM Reference Format:
Abhinand S. Nasari, Hieu T. Le, Richard Lawrence, Zhenhua He, Xin Yang, Mario M. Krell, Alex Tsyplikhin, Mahidhar Tatineni, Tim Cockerill, Lisa M. Perez, Dhruva K. Chakravorty, and Honggao Liu. 2022. Benchmarking the Performance of Accelerators on National Cyberinfrastructure Resources for Artificial Intelligence / Machine Learning Workloads. In *Practice and Experience in Advanced Research Computing (PEARC '22), July 10–14, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3491418.3530772

PEARC '22, July 10–14, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9161-0/22/07…$15.00
https://doi.org/10.1145/3491418.3530772

1 INTRODUCTION

Researchers increasingly depend on specialized hardware for their computing workflows. In 2022, the NSF-funded Accelerating Computing for Emerging Sciences (ACES), a novel CI testbed with accelerators to support scientific ML workflows will be available to the national CI community. Hosted at Texas A&M University, this test-bed complements the GPU-equipped NSF Frontera and NSF-Expanse resources, the Habana-equipped NSF Voyager resource, and the planned FPGA-equipped NSF National Research Platform resource at San Diego Supercomputing Center (SDSC). These NSF investments in CI are strengthened by powerful regional GPU-enabled CI resources such as Lonestar 6 at the Texas Advanced Computing Center (TACC), HiPerGator AI at the University of Florida (UF), and Grace at Texas A&M University. The CI-ecosystem is today in a position where hardware-availability will no longer be a determining factor in CI research-productivity or scope [1]. Our previous work focused on developing platforms that help researchers adopt upcoming CI technologies in their workflows [1]. There have been several proven cases in history where the availability of hardware (and software) slowed down research and development and there are probably much more unknown cases [2, 3]. Deep learning started as early as 1963 [4] but it required the repurposing of GPUs (circa. 2005) to achieve real breakthroughs with AlexNet in the ImageNet benchmark in 2012 [5]. While researchers can explore new techniques on various architectures, there is an urgent need to understand the optimal means to use these accelerators in research CI. To achieve this, we need to understand (a) how to use emergent CI technologies (b) when to use them, (c) the trade-offs between the various technologies and (d) the challenges to reproducibility introduced by various accelerators.

The unique NSF ACES test-bed provides a portfolio of accelerators such as GPUs, Vector Engines, Field Programmable Gate Arrays (FPGAs), and the novel Graphcore IPU accelerators. As such researchers will soon be able to explore the benefits and limitations of GPU- and IPU-accelerated AI/ML workflows. In machine learning, a mathematical model needs to be evaluated with many data elements as the inputs. During training, the model is updated over time to more accurately predict the data elements, offering opportunities to reduce training time by hardware acceleration and parallelization. To develop an understanding of these mechanisms, we need to investigate the most common type of hardware accelerators, the GPU, and a relatively new type of hardware acceleration, the IPU, in the context of machine learning artificial intelligence research. GPUs and IPUs implement Arithmetic Logic Units (ALUs) differently. GPUs are designed to perform floating point operations simultaneously across a large array, with ALUs controlled centrally and sharing memory. In contrast, the ALUs of an IPU are divided into smaller units called tiles, which act independently of one another, having their own local memory and enough compute power to prepare their own instructions. Tiles communicate with each other directly over high-bandwidth channels, even across different IPUs. As such, GPUs are good at certain types of ML models like classical dense Convolutional Neural Networks that require "parallelizing a set of simple decomposable instructions such as matrix multiples" [4]. On the other hand, the graph model of the IPU and its hardware architecture offers opportunities to fine-grained parallelism like for example with small batch sizes or grouped operations like group-convolutions or group-normalizations [11-13] useful in computer vision [14,179,13], natural language processing [18], audio processing, graph neural networks, and Markov Chain Monte Carlo (MCMC) ([19]. In part, this work seeks to illuminate some of the considerations that might lead a researcher to choose one or the other architecture, and how to go about making the transition. By implementing the deep learning framework into the step of force calculation, IPUs have also been shown to leverage the performance of molecular dynamics calculations. Whereas this paper is one of the first to discuss IPU application and its aspect, there is a large corpus of literature on the application of GPUs in applications and its inner workings [6-8].

The richness of the emerging CI landscape presents new opportunities for our research community. Challenges, however, remain for a researcher who wishes to reproduce published results from hardware providers, or apply them to their own research. With the increasing complexity of research workflows, researchers could potentially find that the sub-tasks in a workflow may have to be attributed to different architectures to maximize its performance. These factors are further compounded for reviewers who make recommendations for allocations for AI/ML workflows to national CI resources via mechanisms such as the NSF XSEDE Resource Allocation Committee (XRAC) [9], the NSF Cyberinfrastructure for Sustained Scientific Innovation (CSSI) program [10], the international HPC COVID-19 consortia [11], and NSF Cloud-bank related processes [12]. Indeed, the XSEDE XRAC would benefit from guidance on realistic benchmarking and scaling of AI/ML processes on national CI resources. The standard AI/ML benchmarks are typically optimized for maximum performance and cannot test every scenario the researchers might face. There is no recipe to turn a benchmark result into a useful guide for the researcher's specific workflow. In order to assign workflows optimally on different hardware architectures, it is necessary to perform benchmark tests on representative problems. In this manuscript, we present a student-led benchmarking effort that investigates real-world performance on accelerators at several major CI resources. In subsequent sections of this paper, we describe the methods and computational platforms used to investigate the performance of GPUs and IPUs. We visit these benchmarks on "production" hardware at Texas A&M University, TACC, UF, and SDSC to derive a realistic framework that helps researchers make optimal choices for their workflows. Whereas execution of GPU programs is well understood, we elaborate on how we executed IPU programs and how certain parameters influence the performance to improve the understanding of this new hardware and related software. We analyze the results and discuss our findings in subsequent sections. We finally conclude with recommendations and opportunities for future work. In consideration of the ubiquitous presence and use of GPUs at national CI sites, this paper focuses on giving a better understanding of the new hardware architecture, IPU, and the underlying user interaction experience and considerations for running machine learning workloads.

2 METHODS

2.1 Benchmarking

In this manuscript, we focus on well-known AI/ML models that have been optimized for both GPUs and IPUs, ensuring that we see the absolute best-case scenarios for each. The benchmarks

encompass a variety of models and are intended to be portable to a variety of hardware architectures. The target of a machine learning benchmarking is the time taken to reach a target accuracy when training an algorithm, or latency and throughput when applying the algorithm in real-world inference scenarios, where the algorithm also has to fulfill certain quality requirements on the predictions. This is the fairest comparison between different training methods since it properly accounts for both throughput and accuracy. The optimal training strategy involves selecting the parameters that yield both a high throughput and a high accuracy. This implies that different models will potentially be better suited for one type of hardware architecture or the other, depending on how the training parameters affect the accuracy of that model. We note that hardware and software developers compete to get the best MLPerf™[13] times using a curated list of benchmarks.

For this study, we selected some well-known machine learning models: ResNet [14], EfficientNet [15], MobileNet [16], and BERT models [17]. ResNet, EfficientNet, and MobileNet are classical image classification models built with convolutional neural networks (CNN), and also, they are the backbones of other tasks (e.g., object detection, segmentation) in computer vision. BERT (i.e., Bidirectional Encoder Representation from Transformers) is an important method on many Natural Language Processing (NLP) tasks. These models are supported by both NVIDIA and Graphcore who provide useful examples and optimal parameters. Additionally, we analyze the translation of an application from one accelerator to the other for a custom test model. We especially investigate the influence of the batch size and scaling behavior. In the ideal scenario, we assume perfect linear scaling, i.e., doubling the number of accelerators doubles the throughput and thus cuts the time to train in half. In practice, each application is different, the impact of the batch size on convergence, as well as individual settings of the hardware have to be considered and fine-tuned to achieve close to optimal scaling. In addition, we are interested in the acceleration of the total hardware package, including non-compute resources such as networking and memory.

2.2 Batch size

An important training parameter for ML tasks is batch size, which is the number of training data elements that will be processed before updating the model. The batch size requires delicate tuning because it is relevant for throughput, accuracy, and time to train, i.e., the time to reach a certain accuracy. Time to train balances information of accuracy and throughput: it is quite often possible that the throughput is high, but it takes much longer to reach the accuracy target compared to a setup with lower throughput but reaching a convergence criterion faster. Each model has its own specific expectations for the batch size because it is related to convergence. Updating the model too frequently can cause the model to converge slowly or fail to converge, as the training process is stochastic, and each data element can introduce wild variation and mask the underlying pattern. Similarly, updating the model too infrequently can cause the model to converge slowly or fail to converge, if it happens to get stuck in a local minimum, because any variance in the data gets diluted by the average [18, 19]. However, in addition to the model-specific limitations, there are hardware-specific limitations. If the batch size is too small, then there may not be enough data to keep all the compute resources busy and the compute resources are wasted. If the batch size is too large, then fitting the data in memory and delivering it rapidly to the computing resources becomes a challenge. Thus, for any combination of a model and a hardware architecture, there will be a range of batch sizes which are effective. We observe that the choice of batch size has a measurable impact on throughput and time-to-train, and this relationship is different for different hardware architectures. IPUs offer greater throughput with smaller batch sizes, while GPUs have better throughput performance with larger batch sizes. The absolute ranges for batch size differ substantially. The IPU reaches highest throughput even for small batch sizes, whereas the GPU requires much bigger batch sizes to create large arrays which benefit from its acceleration. The reason for this difference is directly related to the internal details of the hardware architecture designs and are explored in this paper.

2.3 Performing the calculations

NVIDIA and Graphcore container catalogs provide precompiled machine learning software that is optimized for use with their accelerators. They also provide a repository of example machine learning models with optimal parameters for those. Graphcore provides a development toolkit for building software for use on their IPUs. Through the Graphcloud service, Graphcore provides access to IPUs, preinstalled software, and even training data. The methods chosen in this work are based on the information provided by the two vendors and are intended to provide useful advice for a researcher who wished to replicate the procedures.

GPU calculations were performed on NVIDIA A100 (40GB, PCIe)[20] and NVIDIA RTX GPUs [21] on Grace at Texas A&M University. These GPUs are integrated into dual-socket 48-core Intel Cascade Lake compute nodes, with one GPU on each socket. TensorFlow and PyTorch software frameworks are supported by NVIDIA through their container catalog, which is ported to the Grace cluster via the Singularity container runtime. These same frameworks are also built locally on the Grace cluster from source using the EasyBuild framework. This work was expanded to NVIDIA T4 GPUs on Grace and NVIDIA V100 GPUs on NSF Expanse. To ensure that these runs were performed in an optimal manner and comparable to published results, benchmark experiments were performed on standalone NVIDIA A100 GPUs on LoneStar6 at TACC and on NVIDIA A100-equipped DGX servers on NSF Frontera at TACC and HiPerGator AI at the University of Florida. Benchmarking tools for common AI/ML models are also provided by NVIDIA and have been pre-populated with optimal parameters for running those modes on A100 GPUs. The testing environment on the GPU-providing clusters is set up using the Singularity container runtime engine, which enables a common container image to run on different local node architectures. The container images are provided by the NVIDIA container registry [22]. In particular, the nvidia/pytorch:21.11-py3 image for PyTorch and nvidia/tensorflow:20.06-tf1-py3 images were selected for TensorFlow frameworks. Code to perform benchmark tests using these

containerized software stacks is provided by NVIDIA GitHub repository DeepLearningExamples along with tutorials and even some published benchmark results.

For the PyTorch framework, association of GPUs is handled by NVIDIA's CUDA framework, which automatically detects and selects local available GPUs. Distribution of training tasks across multiple GPUs is handled within the PyTorch framework, which internally controls its processes. Specification of the number of processes is done as an argument to the training script. The training script is executed within the container.

For the TensorFlow framework, association of GPUs is handled by NVIDIA's CUDA framework, which automatically detects and selects local available GPUs. Distribution of training tasks across multiple GPUs is accomplished using the mpirun command, which has several important arguments. For example, the command below starts two processes, each of which will associate itself to a different GPU, because they are located on different sockets. The NVIDIA TensorFlow ResNet 50 benchmarking example is executed within the container and within an mpirun launch, if applicable. It is important to note that batch size and the precision (tf AMP) are both selected at this stage, whose values are specific to a GPU type. While this example is for the A100 GPU, scripts for other GPUs were similarly implemented [23].

Calculations were performed on Graphcore IPU-POD16 compute nodes running with a Dell R6525 server on the Graphcloud. Graphcloud is equipped with a large number of Graphcore Colossus GC200 IPUs (also referred to as Mk2 IPUs). 4 such IPUs are located in each IPU-M2000 machine, and four such machines are integrated into the virtual IPU-POD16 compute node with 8 NUMA nodes. Whereas the experiments were performed with the classic systems, TensorFlow and PyTorch software frameworks are supported by Graphcore through their Poplar software development kit. Benchmarking tools and documentation for common AI/ML models are also provided by Graphcore and offer a detailed description of running environments [24, 25]. Here, we briefly discuss some salient factors. Environment variables are set to use the Graphcore tools and Poplar, a graph programming framework for IPUs. Parallel host processes on the IPU-POD16 (number of instances) for data loading and preprocessing are launched using the "poprun" command while specifying the number of replicas (model copies) where, each replica is an identical copy of the same graph. Graphs can be partitioned across a number of IPUs working on different subsets of that graph, or each IPU could keep the entire graph in memory. The total number of IPUs equals the number of IPUs per replica multiplied by the number of replicas. We specify the number of processes running on the host CPUs. These instances handle data input/output and are not responsible for the computation on the IPUs. This value is chosen based on how intense the data load is expected to be for the model. It is generally equal to or a divisor of the number of replicas.

3 RESULTS

A series of benchmark experiments were performed on GPUs and IPUs as described above. Owing to different CPU technologies and software stacks, we anticipated small differences in performance on the A100 GPUs on Grace, Lonestar 6, HiPerGator AI at the University of Florida. The runs on Expanse were performed on NVIDIA V100 GPUs. Performance on NVIDIA GPUs was checked using common methods like NVIDIA SMI for GPUs and the PopVision Graph Analyzer for Graphcore IPUs. PopVision Graph Analyzer is a visualization tool that summarizes how a program is performing on the IPUs. Features include Summary, Memory Report, Liveness Report, and etc. The Liveness Report displays the memory usage throughout the lifetime of the program, showing the operations or layers of the model using the most memory (e.g., the spikes in the graph below). Based on these data, one can optimize the model.

One of the key considerations of this work is to understand how performance scales with available resources. Here, we examined the scaling efficiency of the GPUs and IPUs using the ResNet50 V1.5 model benchmark experiment on PyTorch and TensorFlow. This is a convolution neural network (CNN) that performs image classification [14]. This model is trained using image data from the ImageNet 2012 machine learning challenge [26]. ResNet50 training requires a lot of data (ImageNet2012) and data loading and preprocessing is a burden. Since a lot of optimizations should be on the host side, a distributed-friendly dataset can be generated to reduce read times from disk. This is the webdataset format dataset for PyTorch and tfrecord format dataset for TensorFlow. As measurement of performance, we us throughput, measured in images per second. A reasonable batch of images for training ResNet 50 is on the order of a hundred, and a single batch of images can be processed in parallel efficiently. Thus, an excess of file read operations can be a bottleneck if not handled properly. For the case of PyTorch, Graphcore supports a tape-archive based file format that reduces the number of file read operations to an acceptable level. PyTorch with NVIDIA must be treated carefully so as to minimize the effect of the file read bottleneck. For the case of TensorFlow, both Graphcore and NVIDIA support a TensorFlow-specific file format that reduces file read operations to an acceptable level. Keeping the data on the closest local disk to the GPU helps, but a slight reduction in performance is still expected.

3.1 PyTorch ResNet50 Scaling

As described in previous sections, the experiments on IPU POD-16 were performed using the recommended ratio of one instance per replica. Table 1 describes the effect on scaling the number of IPUs from 1 to 16 in an incremental manner. The batch size parameter in the benchmarking code specifies the per-IPU minibatch. For measuring the increase in throughput due to the increase in resources, training is measured after a fixed number of epochs and only the throughput and training time for those first few epochs is recorded. This is represented in the data as we see that the throughputs in the first and second epochs are significantly different, especially when the IPU count is larger. This is because in the first epoch, the graph compilation is included. Considering the throughput for the second epoch allows us to focus on the part of the performance which is due to hardware acceleration and not due to the training method. As such it is more representative when evaluating performance in this study. We note a performance drop when the number of instances exceeds the number of NUMA nodes on the pod system. On starting more than 8 instances, the processes are dedicated to

Table 1: ResNet50 model Training performance on various number of IPUs with different instances setup

Total IPU Count	Instances	Replicas	Throughput (images/sec)	
			1st epoch	2nd epoch
1	1	1	1,113.0	1,862.9
2	2	2	1,538.2	3,717.2
4	4	4	1,893.9	7,078.1
8	8	8	2,233.7	13,931.7
16	8	16	2,189.7	23,082.9
16	16	16	2,256.2	18,532.6

the NUMA nodes and are swapped among them. As such, we find that for the POD-16 with 8 NUMA nodes, the best performance was achieved with a configuration of 8 instances as opposed to 16 instances. We find that when the IPU count is small (e.g., ≤8), the training performance (image throughput) scales almost linearly with the IPU count. However, when the IPU count continues to increase, the performance starts decreasing. The last entry in the table shows a performance drop because the number of instances exceeds the number of available NUMA nodes, and the processes swap among the nodes, overwhelming the host server.

Similar experiments were performed on NVIDIA GPUS. NGC containers were obtained from the NVIDIA container repository using standard processes identified by NVIDIA [27]. These were run on several CI sites to afford benchmarks for NVIDIA T4, V100, A100, and DGX-A100 GPUs. The PyTorch ResNet50 training performance benchmarks on GPUs were calculated with both TF32 and mixed precision, respectively. The number of GPUs were incrementally increased. As shown in Figure 1, our benchmark experiments on the DGX-A100 GPUs were largely in agreement with figures published by NVIDIA, establishing the veracity of our approach. A potential source of discrepancy could be the fact that our runs were performed on DGX-A100 servers equipped with 40 GB of memory as opposed to the NVIDIA official runs that were performed on DGX-A100 servers equipped with 80 GB of memory. As anticipated, we note that the A100 GPUs outperform the T4 and V100 GPUs in these experiments. We note that the stand-alone A100 GPUs on Intel Cascade Lake (Grace Cluster) and on the AMD Milan processors (Lone Star6) are less performant than the comparable DGX-A100 servers. Irrespective of the platform, we find that the scaling performance of GPUs and IPUs show similar magnitude changes per-device. We note that a single IPU is not directly comparable to a single GPU. On the CirraScale cloud, we note that IPUs are priced differently from A100 GPUs. [1] For energy consumption, one notes that a single Colossus Mk2 IPU has a TDP of 300 W. An A100 DGX with 8 GPUs has a system TDP of 6500 W[2], whereas an IPU-POD16 consists of 16 IPUs with a TDP of 6000 W[3] plus the TDP of the host which can vary due to host disaggregation.

[1]https://cirrascale.com/cirrascale-cloud-pricing.php
[2]https://images.nvidia.com/aem-dam/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf
[3]https://docs.graphcore.ai/projects/graphcore-ipu-m2000-datasheet/en/latest/product-description.html

3.2 Additional PyTorch model results

In addition to ResNet 50, we also ran several popular models using the PyTorch framework on IPUs. GPU performance numbers have been reported by NVIDIA. Similar runs of Efficient Net B0 on A100 and V100 GPUs found throughputs of 3749.09 img/s (2 A100s) and 5745 img/s (4 V100s). Here, we used the same approach as in our previous runs but replaced the respective configuration file as suggested in the Graphcore public examples for PyTorch CNN models.

3.3 TensorFlow ResNet50 Scaling

3.3.1 TensorFlow ResNet50 on IPUs. When training ResNet50 with TensorFlow on IPUs for the purpose of computing throughput, only 1 epoch is used, because graph optimization occurs before epoch 1 in the Poplar variant of TensorFlow. For the number of IPUs used in this work, the batch size is selected to be small (20), to fit in memory on as little as a single IPU. To effectively increase the batch size for convergence considerations, the gradient accumulation feature is used with a value of 24. Batch norm quality can improve with aggregating statistics over a large number of samples, which is achieved on the IPU by distributing batch norm across 2 IPUs (BN_spann) This yields a total effective batch size of 20 x 24 or 480 per IPU for gradient updates and a micro-batch size of 40 for normalization purposes. A benefit of the gradient accumulation is that one can reduce the gradient accumulation when increasing the number of IPUs, so that the number of images assigned to a single gradient update (effective batch size) at one time remains roughly constant. Thus, hyperparameters do not have to be reoptimized when scaling the application horizontally to more IPUs. The results from our experiments on the ResNet model are described in Table 3

We first investigate the effect of batch size on a single IPU. For the case of the single IPU run, the micro batch size was varied from very small (2) to optimal (20) to investigate the lower bound of efficiency. ResNet 50 with TensorFlow-1.15 with mixed precision, approaches its maximum value quickly at small batch sizes, with only a small performance penalty for moderately small micro batch sizes and a maximum batch size of 20. Greater batch sizes for convergence considerations can always be achieved by delaying the gradient accumulation, so the micro batch size forms a lower bound for effective batch sizes. The lower bound for the IPU batch size is an advantage for researchers using models that are optimal at small batch sizes. On investigating the scaling of throughput across one or more IPUs, throughput increases with increasing batch sizes. We find almost linear scaling as one increases the number of IPUs.

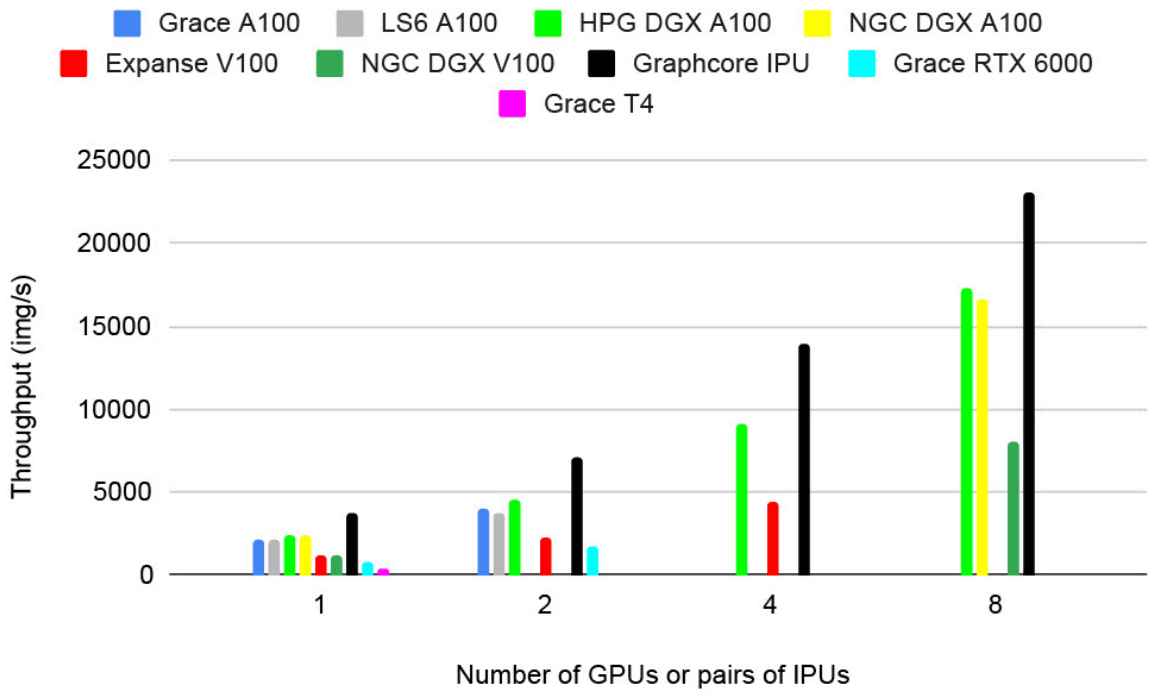

Figure 1: Comparison of GPUs and IPUs on the ResNet 50 model in PyTorch with mixed precision. Scaling is similar for GPUs and IPUs, with similar magnitude per-device.

Table 2: Training performance of different models on various number of IPUs

Model	IPU/replica	Replicas	Throughput(img/sec or items/sec)
Efficient Net B0	2	2	8,905 img/sec
Efficient Net B4	4	4	4,200 img/sec
MobileNet v3 small	1	4	7,222.6 img/sec
MobileNet v3 large	1	16	7,347.6 img.sec
BERT Base Squad fine tuning	4	1	1,297 items/sec

Results from other runs (non-optimal) for exploring the parameters and a discussion are offered elsewhere [23] .

3.3.2 Establishing Relevant Reference Benchmarks. Results from benchmark experiments of the ResNet50 model, developed with TensorFlow 1.15, on GPUs are shown in Figure 2 Much like our runs with PyTorch, NVIDIA A100, DGX A100, T4, and V100 GPUs were selected from multiple sites, with the same model and ImageNet dataset, and the same number of epochs (one). Batch size is 256 on the A100, V100, RTX 6000, and T4 when using mixed precision. We note that these runs are under optimal conditions to maximize performance on these architectures. NVIDIA mixed precision is not identical to Graphcore mixed precision, but the most comparable option available in this case. The results are similar to those observed in our experiments with PyTorch. The runs match NVIDIA's published benchmarks for DGX-A100s. We see that the IPUs performance is closer to that of A100 GPUs as opposed to the NVIDIA T4 and RTX GPUs.

3.4 Custom Test Model on TensorFlow

Over the past decade, ML-based image compression algorithms have achieved great improvement, surpassing standard codecs, JPEG, JPEG2000, WebP, BGP in both visual quality and standard metrics. Computational cost, ease of use, and effective entropy models are areas of active research. In this benchmark, a representative image compression model that resembles an Autoencoder is created. This model consists of residual blocks and one bottleneck. It simplifies the architecture of the compression model, which has many additional bottlenecks and compression mechanisms to quantize the bottlenecks. Though simplified, the model maintains the structure of the actual compression model. The model is built on

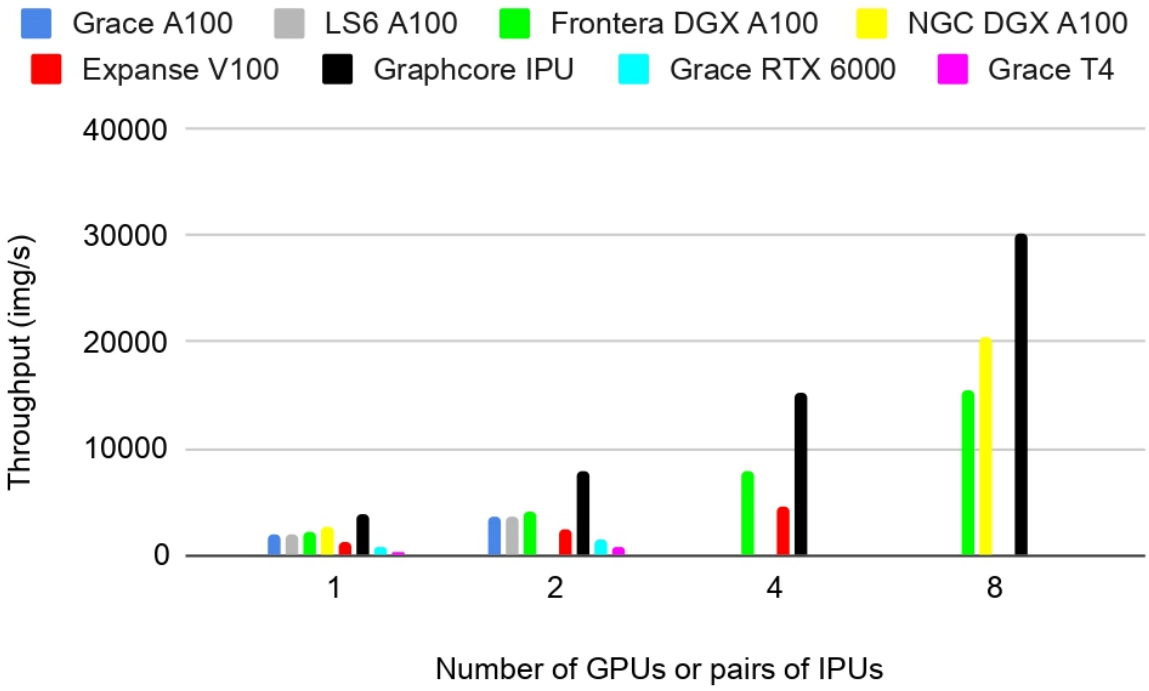

Figure 2: Comparison of GPUs and IPUs on the ResNet 50 model in TensorFlow-1 with mixed precision. Scaling is similar for GPUs and IPUs, with similar magnitude per-device.

Table 3: Performance data for scaling in IPU runs. Model was ResNet 50 with TensorFlow 1.

BN_span	Batch size	num-instances	num-replicas	Total IPUs	Throughput (images/sec)	Training Time (sec.)
1	2	1	1	1	786	1,948.5
1	4	1	1	1	1,181	1,297.3
1	8	1	1	1	1,608	952.8
1	14	1	1	1	1,884	813.0
1	16	1	1	1	2,030	754.5
1	18	1	1	1	2,033	753.6
1	20	1	1	1	2,130	719.4
2	20	1	2	2	3,895	393.4
2	20	2	4	4	7,804	196.4
2	20	4	8	8	15,274	99.7
2	20	8	16	16	30,170	50.5
2	20	32	64	64	107,535	11.8

IPU-TensorFlow 2.4 and trained on the MNIST dataset, to reconstruct any given images. In essence, it is similar to the ResNet50 model and can run on both IPUs and GPUs, allowing for a fair comparison between the two hardware platforms. When training with IPUs, the batch size is set to 10, because it is the maximum batch size that the model can fit into IPUs' internal memory. The performance is determined by throughput, which is the total number of images per second. We note that different training configurations lead to different total number of training images, because the number of images need to be preprocessed to be compatible with the set-up of each model. To keep all values consistent, time spent on each epoch is scaled to match the time spent on training the original number of images, which is 70,000 images.

Results from these experiments are shown in Table 4. We find that when using the same small batch size (10 samples/step), throughput of the model trained on IPUs (around 3000 images/second) is much

Table 4: Performance of the representative model on the Mk2 IPU-POD16 and A100-GPUs

Hardware Platform	Replicas (Devices)	Micro batch size	Time spent on 2nd epoch (seconds)	Throughput(images/second)
A100 GPU	2	512	8.1	8,642
A100 GPU	2	256	11.7	5,983
A100 GPU	2	128	21.8	3,211
A100 GPU	2	10	255.5	274
A100 GPU	1	10	259.0	270
Mk2-IPU	1	10	23.7	2,954
Mk2-IPU	2	10	13.5	5,185
Mk2-IPU	4	10	9.0	7,743
Mk2-IPU	8	10	6.3	11,052
Mk2-IPU	16	10	4.9	21,547

larger than the throughput of the other model trained on GPUs (270 images/second). In contrast, we see tremendous speed up on GPUs as we increase the batch size. The reason behind this is that the IPUs distribute workloads on tiles. The NVIDIA A100 has much larger on-board memory, which means that it can handle a much larger batch size. Particularly, the batch size of the GPU model can reach 512 samples/step. This leads to a great improvement in training performance on the GPU, whose throughput is 8,642 images/second at a batch size of 512. It is noted that the model is partially optimized for the IPUs. In the case of the 16-IPU model, after implementing the gradient accumulation method and distributed data loading with Poprun, the throughput of the model increases from 14,000 to around 21,500 images/second, offering almost linear scaling.

4 CONCLUSION

In this student-led study we successfully investigated the performance of accelerators on several national CI resources. Working with the project teams, we further queried the effect of batch size on the throughput of AI/ML runs. In agreement with the expectations put forward by previous studies, we have observed that the performance of ML workflows is highly sensitive to the distributed workflow configuration. Future researchers need to carefully set up the configuration to achieve optimal performance. A small deviation from the optimal configuration can lead to a massive drop in performance, to the point of wasted accelerator resources. Although most optimization happens under the hood by the compilers, there are a few knobs to tweak optimal performance. Usually, those can be divided into optimizations for memory, communication, and compute efficiency/speed but there is a lot of cross-interaction. We find that increasing the batch size increases the memory pressure and the amount of data that needs to be communicated, but most importantly it usually improves hardware utilization and thus computation on IPUs for smaller batch sizes and for GPUs on larger batch sizes. Whereas the impact of batch size on throughput is rather straightforward and easy to tune, the influence of the batch size on accuracy is more subtle, and will be the subject of a future study. With very small batches, the estimate of the gradient will be too noisy and the algorithm might not converge, converge slowly, or not reach a target accuracy. This is compensated on the IPU with gradient accumulation. With very large batch sizes, a lot of data needs to be processed before the weights get updated, which can significantly slow down convergence and might require to actually work with smaller batch sizes and less hardware utilization, especially when scaling applications. Hence, depending on the application and settings at hand, IPU or GPU can be the preferred choice depending on the requirements on the choice of the batch size as well as the resulting throughput. It is thus vitally important that the CI community provides consistent comprehensible information to researchers interested in using accelerator resources. Part of this is to ensure that the distribution frameworks provide accurate configuration suggestions, and part of this is to maintain a standard of benchmarking so that researchers can easily identify poor quality results. Under the assumption that this information is readily available, it is certain that most researchers would find that IPUs or GPUs are of massive utility in their research. We note that new GPU and IPU architectures would be available for both GPUs and IPUs. Graphcore's more recent BOW-POD 16 will be available to researchers nationwide on the upcoming NSF ACES testbed.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) award number 2112356 ACES - Accelerating Computing for Emerging Sciences, NSF award number 1925764 SWEETER - SouthWest Expertise in Expanding, Training, Education and Research, NSF award number 2019136 BRICCs - Building Innovation at Community Colleges, NSF award number 2019129 FASTER - Fostering Accelerated Scientific Transformations, Education, and Research, Graphcore, staff at Texas A&M HPRC, Chris Peter Francis, and Dr. Jian Tao. We would like to thank the University of Florida and Erik Deumens for providing an allocation on the HiPerGator AI HPC system.

REFERENCES

[1] Michael Lau, Stuti Trivedi, Zhenhua He, Tri Pham, Lisa Perez, and Dhruva Chakravorty. 2021. Research Cloud Bazaar: A software defined cloud workflow cost management tool. In Practice and Experience in Advanced Research Computing, ACM, Boston MA USA, 1–4. DOI:https://doi.org/10.1145/3437359.3465602
[2] Sara Hooker. 2020. The Hardware Lottery. arXiv:2009.06489 [cs] (September 2020). Retrieved February 16, 2022 from http://arxiv.org/abs/2009.06489
[3] Ray Kurzweil. 1990. The age of intelligent machines. MIT Press, Cambridge, Mass.
[4] K. Steinbuch and U. A. W. Piske. 1963. Learning matrices and their applications. IEEE Trans. Electron. Comput. EC-12, 6 (December 1963), 846–862. DOI:https://doi.org/10.1109/PGEC.1963.263588

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (May 2017), 84–90. DOI:https://doi.org/10.1145/3065386

[6] Martin Burtscher, Rupesh Nasre, and Keshav Pingali. 2012. A quantitative study of irregular programs on GPUs. In 2012 IEEE International Symposium on Workload Characterization (IISWC), IEEE, La Jolla, CA, USA, 141–151. DOI:https://doi.org/10.1109/IISWC.2012.6402918

[7] D. Steinkraus, I. Buck, and P.Y. Simard. 2005. Using GPUs for machine learning algorithms. In Eighth International Conference on Document Analysis and Recognition (ICDAR'05), IEEE, Seoul, South Korea, 1115-1120 Vol. 2. DOI:https://doi.org/10.1109/ICDAR.2005.251

[8] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P. Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. arXiv:1804.06826 [cs] (April 2018). Retrieved February 18, 2022 from http://arxiv.org/abs/1804.06826

[9] The Extreme Science and Engineering Discovery Environment (XSEDE). 2021. Retrieved from https://www.xsede.org/

[10] National Science Foundation. 2021. Retrieved from https://beta.nsf.gov/funding/opportunities/cyberinfrastructure-sustained-scientific-innovation-cssi

[11] COVID-19 HPC Consortium. 2021. Retrieved from https://covid19-hpc-consortium.org/

[12] National Science Foundation. 2021. Retrieved from https://www.cloudbank.org/

[13] MLPerf, https://github.com/mlcommons/, 2022. The MLPerf name and logo are trademarks of MLCommons Association in the United States and other countries. All rights reserved. Unauthorized use is strictly prohibited. See www.mlcommons.org for more information. Results reported are not verified by the MLCommons™ Association.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, 770–778. DOI:https://doi.org/10.1109/CVPR.2016.90

[15] Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946 [cs, stat] (September 2020). Retrieved February 18, 2022 from http://arxiv.org/abs/1905.11946

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs] (April 2017). Retrieved February 18, 2022 from http://arxiv.org/abs/1704.04861

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs] (May 2019). Retrieved February 18, 2022 from http://arxiv.org/abs/1810.04805

[18] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. arXiv:1609.04836 [cs, math] (February 2017). Retrieved February 16, 2022 from http://arxiv.org/abs/1609.04836

[19] Dominic Masters and Carlo Luschi. 2018. Revisiting Small Batch Training for Deep Neural Networks. arXiv:1804.07612 [cs, stat] (April 2018). Retrieved February 16, 2022 from http://arxiv.org/abs/1804.07612

[20] NVIDIA. 2020. A100 40GB PCIe Product Brief. (September 2020). Retrieved from https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf

[21] NVIDIA. 2022. NVIDIA RTX. Retrieved from https://www.nvidia.com/en-us/design-visualization/rtx/

[22] NVIDIA. 2022. NGC catalog. Retrieved from https://catalog.ngc.nvidia.com/containers

[23] https://github.tamu.edu/HPRC/Graphcore_IPU_Benchmarks

[24] TAMU HPRC Wiki. 2021. Retrieved Feb 18, 2022 from https://hprc.tamu.edu/wiki/

[25] Graphcore documents. 2022. Retrieved Feb 18, 2022 from https://docs.graphcore.ai/en/latest/

[26] 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. arXiv:1409.0575 [cs] (January 2015). Retrieved February 17, 2022 from http://arxiv.org/abs/1409.0575

[27] NVIDIA. 2022. NGC catalog. Retrieved from https://catalog.ngc.nvidia.com/containers