Insight Gained from Migrating a Machine Learning Model to Intelligence Processing Units

Hieu Le
hieult@tamu.edu
Texas A&M University
College Station, TX, USA

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

Mai Le
University of Houston
Houston, TX, USA
mnle8@cougarnet.uh.edu

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

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

Akhil Chilumuru
Texas A&M University
College Station, TX, USA
akhilchilumuru@tamu.edu

Yan Yao
University of Houston
Houston, TX, USA
yyao4@central.uh.edu

Jiefu Chen
University of Houston
Houston, TX, USA
jchen82@central.uh.edu

ABSTRACT

The discoveries in this paper show that Intelligence Processing Units (IPUs) offer a viable accelerator alternative to GPUs for machine learning (ML) applications within the fields of materials science and battery research. We investigate the process of migrating a model from GPU to IPU and explore several optimization techniques, including pipelining and gradient accumulation, aimed at enhancing the performance of IPU-based models. Furthermore, we have effectively migrated a specialized model to the IPU platform. This model is employed for predicting effective conductivity, a parameter crucial in ion transport processes, which govern the performance of multiple charge and discharge cycles of batteries. The model utilizes a Convolutional Neural Network (CNN) architecture to perform prediction tasks for effective conductivity. The performance of this model on the IPU is found to be comparable to its execution on GPUs. We also analyze the utilization and performance of Graphcore's Bow IPU. Through benchmark tests, we observe significantly improved performance with the Bow IPU when compared to its predecessor, the Colossus IPU.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

ACES (Accelerating Computing for Emerging Sciences), Graphics Processing Unit, ResNet50, Intelligence Processing Unit, Classification, Prediction, Convolution Neural Network, Optimization

PEARC '24, July 21–25, 2024, Providence, RI, USA

ACM ISBN 979-8-4007-0419-2/24/07
https://doi.org/10.1145/3626203.3670527

ACM Reference Format:
Hieu Le, Zhenhua He, Mai Le, Dhruva K. Chakravorty, Lisa M. Perez, Akhil Chilumuru, Yan Yao, and Jiefu Chen. 2024. Insight Gained from Migrating a Machine Learning Model to Intelligence Processing Units. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3626203.3670527

1 INTRODUCTION

Scientists are increasingly relying on specialized hardware to facilitate computationally-intensive research. By 2022, the U.S. National Science Foundation (NSF)-funded Accelerating Computing for Emerging Sciences (ACES) offered the national community a cyberinfrastructure (CI) testbed equipped with a range of accelerators that were developed for artificial intelligence and machine learning (AI/ML) workflows. Hosted by Texas A&M University, ACES complements the NSF ACCESS FASTER high-performance computing (HPC) system that is also hosted by Texas A&M University.

Included in ACES' suite of accelerators are 16 Graphcore Colossus Intelligent Processing Units (IPUs) and 16 Graphcore Bow IPUs. Although IPUs were introduced to the ML community much later than GPUs, Graphcore hardware has shown promising results that are on par with GPU performance for their training and inference capabilities [11], [12].

GPUs and IPUs are designed with different Arithmetic Logic Units (ALUs), which gives each their own advantages depending on the ML workload. GPU ALUs centrally control arithmetic operations and memory sharing. As a result, large-memory GPUs demonstrate superior performance when executing tasks that involve parallelizing a set of decomposable instructions, such as matrix multiplication [14]. Meanwhile, an IPU partitions ALUs into numerous smaller tiles, each of which is independent from the other. Moreover, each tile has its own memory and can compute instructions independently. Thus, fine-grained parallelism is one of the main advantages of IPU architecture.

Recently, a new generation of Graphcore hardware, the Bow IPU, has been introduced. The Bow IPU represents an enhanced

generation of the existing Graphcore chip known as the Colossus Mk2. While maintaining the same number of processing cores and memory capacity as the Colossus Mk2, the Bow IPU achieves up to 40 percent greater performance due to its utilization of a novel wafer-on-wafer design. This innovative technology enables the integration of two distinct silicon wafers in a vertical, or three-dimensional, arrangement to create a unified chip [3].

This paper not only aims to compare the performance of the new Bow-IPU architecture with previous IPU generations, but also investigates the process of transitioning ML models between GPUs and IPUs. Through integrating deep learning frameworks into IPU with careful steps of implementation, IPUs have demonstrated their ability to enhance the performance of ML computations. In contrast with an extensive corpus of literature regarding the utilization and internal mechanisms of GPUs in various applications [2, 8, 15], this paper is among a limited number that discuss the performance of the Bow-IPU and the process of porting ML models from GPUs to IPUs [11, 12].

We show that IPUs offer an alternative to GPUs for scientific research, especially in the field of battery science. As in Fujimura's work, ML is utilized to predict the ionic conductivity of numerous conductors [4]. A recent collaboration between Microsoft and the Pacific Northwest National Laboratory takes a significant stride, utilizing AI to assess more than 32 million solid-state electrolyte candidates within a week which greatly accelerates the process of discovery toward better rechargeable batteries [1]. With increased demand for computing resources, many complex problems can be solved by using the IPU accelerator. With the computing power of IPUs, we successfully trained a specialized mode [9] to predict effective ionic conductivity of composite cathode solid-state batteries. The performance, particularly the training throughput of the model trained on IPUs, is similar to models trained on GPUs.

In the following sections, we outline approaches and computational platforms utilized to examine the performance of the Bow-IPU. While the execution of GPU programs is widely comprehended, we elaborate on the execution of IPU programs and analyze how several implementations impact performance when porting a GPU model to the IPU. Given the widespread use of GPUs at national CI sites, this paper aims to enhance the understanding of the new Bow-IPU architecture and how users can efficiently use ML workflows with IPU accelerators.

2 METHODS

2.1 Benchmarking

In this work, AI/ML models were tested with both GPUs and IPUs. All models were optimized for each platform architecture to guarantee that peak performance could be achieved. The objective of an ML benchmark is to measure the time required to achieve a specific level of accuracy during training, or to assess the speed and efficiency of algorithmic performance in real-world inference situations, focusing on latency and throughput. This comparison ensures fairness among various training methods by appropriately considering both throughput and accuracy. The most effective training strategy entails choosing parameters that produce both high throughput and high accuracy. Therefore, some models may be more compatible and more performant with specific hardware architectures than others, depending on how training parameters impact the model's accuracy.

We performed benchmarks on the Bow-IPU architecture using a couple of ML models. All models were implemented with the core network being the convolutional neural network (CNN), which is also the backbone for many ML tasks such as object detection and segmentation in computer vision. One of many models is the well-known ResNet [5] model which contains different skip connection layers to mainly combat the vanishing gradient problem. This model is heavily optimized for both accelerators to achieve peak performance.

On the other hand, we also analyzed the process of porting a specialized model from GPU to IPU. Besides adding simple wrappers around regular ML modules to make it run on IPU, we addressed the optimization with IPU to fully utilize the hardware for training. Additionally, several advanced options, such as pipelining and gradient accumulation, were investigated to boost IPU performance.

Theoretically, we desired to achieve linear scaling, meaning that if the number of accelerators doubles, the throughput also doubles, thereby reducing the training time by half. However, in practice, doubling the number of accelerators does not result in halving the training time. This is due to various conditions of hardware and non-compute resources, such as memory and networking. Moreover, we are much more interested in the performance of the overall hardware system than individual components. Therefore, many attributes of ML models in both IPU and GPU environments are fine-tuned to achieve the best possible optimization.

2.2 Specialized Model

The escalating demand for safer and more efficient battery technologies has spurred a significant shift towards non-flammable solid-state batteries, and away from their lithium-ion counterparts (which can ignite or explode). While solid-state batteries have safety and energy density advantages, a notable challenge persists in the form of low power density due to non-optimal cathode microstructures that lead to high tortuosity and poor ionic conductivity [7]. High Ionic conductivity corresponds to low internal resistance, resulting in more efficient ion transport. This high ionic conductivity allows fast movement of ions within the composite cathode, enabling higher actual capacity at fast charging and discharging rates, eventually contributing to higher cell energy density, sustained stability, and consistency of performance over numerous charge and discharge cycles. To study the effect of cathode microstructure on ionic conductivity, a general workflow was proposed to predict the ionic conductivity from scanning electron microscopy (SEM) images [10]. The general workflow consists of two steps: image denoising for data preparation and effective conductivity prediction. Cross-sectional SEM images often contain striped noise from ion beam polishing, which impacts the accuracy of conductivity calculations. The denoising model, U-net, effectively eliminates striped noise from SEM images and converts them into clean binary images. The binary images have two regions: conductive (yellow) and non-conductive (black) regions, corresponding to solid electrolyte and cathode active material, respectively. The effective conductivity prediction model is a CNN implemented in PyTorch as shown in Figure

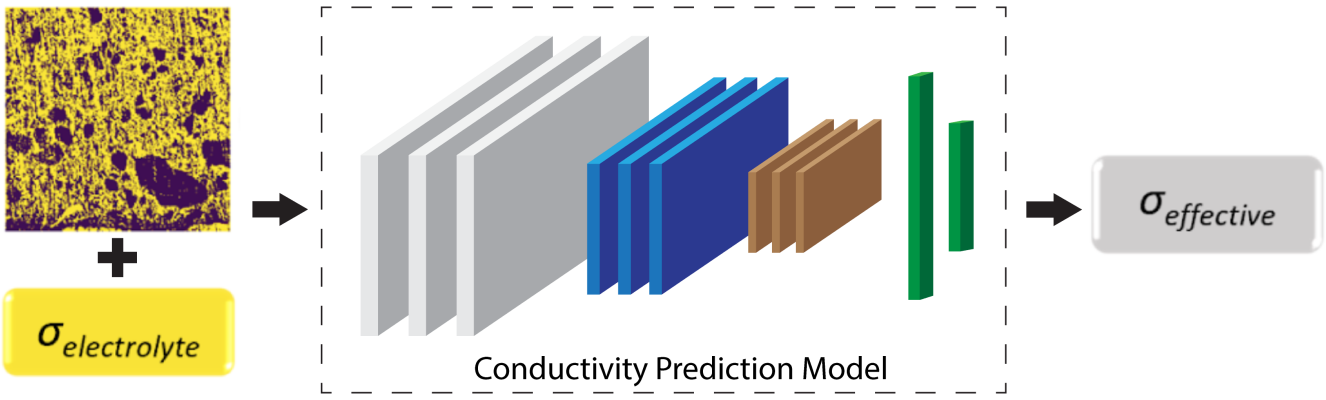

Figure 1: The specialized model for effective conductivity prediction

1. The model takes binary SEM images and the ionic conductivity value of the conductive region as inputs to generate predictions for effective or total ionic conductivity. The conductivity model is the model that we aim to port to the Bow-IPU.

The image datasets consist of training, validation, and testing sets, distributed in a ratio of 1216:152:152. Binary images of size 256x256 were obtained by denoising and cropping SEM images. The reference effective conductivity value of each image is determined using the finite difference method (FDM). FDM is a highly accurate method for calculating current, resistance, and effective conductivity. However, FDM accrues significant computational cost, especially in situations involving large-sized images or when processing a substantial quantity of images. In the work of Mai, et al., FDM was employed to calculate the effective conductivity value for each image, serving as the target value of the effective conductivity prediction model [9]. The acquired image datasets, along with their corresponding effective conductivity values, were utilized to train the CNN effective conductivity prediction model. The model allows a fast prediction of a composite cathode's performance from its SEM images by outputting the predicted effective ionic conductivity value.

2.3 Porting the Conductivity Model to IPU

Different hardware architectures require different techniques and workflows to improve the performance of an ML model. The specialized model is originally implemented to run on GPUs. Thus, when porting to IPUs, several modifications are required to make our model run optimally. Fortunately, the amount of work to modify our workflow is minimal and can be done quickly.

IPU Options

The first modification that we perform is to add Poptorch options to our program. Graphcore provides a placeholder for many options that can be defined for the hardware; thus all options are wrapped in a unique and central variable. IPU options define how IPUs control the workflow of the training and inference of a model. One of the many important options is the device iterations option. The option organizes batches of data on a queue that can be used directly by the IPU, which significantly reduces the time spent on data transferring between CPUs and IPUs. Another advantageous attribute involves instructing the IPU to execute convolution and matrix multiplication in float16, thereby reducing the computational workload, as these two operations account for the majority of computations.

Table 1: Configuration for IPU pipelining

Number of IPUs	Configuration
2	[6, 4]
4	[5, 1, 1, 3]
8	[1, 1, 1, 1, 1, 1, 1, 3]

Dataloader

For implementation on IPUs, we opt for a dataloader specifically designed for IPUs, as opposed to utilizing PyTorch's dataloader. The IPU dataloader provides the flexibility to incorporate additional options for modifying data processing methods. With the predefined Poptorch options mentioned above, we simply add the unique option variable to the IPU dataloader. As a result, we achieve an efficient data pipeline for IPU training and inference processes.

Pipelining

Pipelining is an effective method that we use to improve the performance of our model. Pipelining involves dividing the entire model into several computational stages, with each stage's output being fed into the next. Each stage is allocated an IPU to perform its forward and backward propagation. By distributing the stages across various IPUs and ensuring an adequate number of mini-batches, all stages will concurrently process a data batch after an initial "ramp-up" phase. These stages execute concurrently across multiple IPUs, which enables a more efficient utilization of resources compared to relying solely on sharding. As a result, this pipeline maximizes the parallel processing capabilities of all IPUs involved, leading to enhancements in processing efficiency, throughput, and latency performance.

When employing pipelining with a model on the IPU, it is necessary to specify the gradient accumulation option which enables concurrent processing of several mini-batches through the pipeline, enabling IPUs to execute multiple stages simultaneously. Particularly, in the backpropagation, gradients accumulate across multiple

mini-batches. After processing the predefined number of accumulation, the model parameters are updated with the average value of gradients. With gradient accumulation enabled, the global batch size is defined to be equal to the multiplication of the mini-batch size, the number of replicas, and the number of gradient accumulation.

Graphcore provides several methods to pipeline a model. All of them are equally efficient and the choice of which method to use depends on the user's preference. Using the provided 'poptorch.BeginBlock' wrapper, we are able to control the partitioning of the model's granularly. The process of organizing blocks into different stages is straightforward, which only requires access to components of an ML model.

To further boost the performance of multiple IPUs via pipelining, the amount of workload and memory of each partition should be equally distributed. It is essential to consider that these partitions will be allocated and executed across multiple IPUs. Thus, the way in which we segment a model will have a direct influence on both memory usage and performance at each stage. Subsequent compromises must be taken into account:

- Stages must be accommodated within memory, taking into account both active and intermittently active memory. A report from the PopVision tool can assist with identifying which Ops and tensors utilize the most memory.
- Stages should exhibit comparable execution durations to prevent IPUs from remaining inactive while others finish the computation. The time taken by the longest stage dictates the duration of a single step. Again, the PopVision execution trace can aid in identifying stages with imbalances.
- Communication between IPUs incurs a slower rate compared to accessing memory within a single IPU and should be kept to a minimum.

Even though Graphcore provides convenient methods to pipeline a model, the task still requires manually partitioning the original blocks of a model into a corresponding pipelined block. This step requires a good understanding of all components of the model. The conductivity model is built upon many layers. Based on the model implementation, we group all layers into 10 different blocks, each of which contains a different number of parameters. After analyzing the memory usage and computational workload during pipelining with PopVision, we obtain configurations for pipelining with a different number of IPUs as shown in Table 1.

As can be seen from Table 1, there is no single IPU model. The reason behind this is that the model cannot fit into a single Bow-IPU with an input size of 256x256. This is due to the limited memory of each IPU in the system. To overcome this problem, pipelining is the method Graphcore recommends. Therefore, we came up with three pipelined models:

The number in the second column of Table 1 indicates how many layers are being placed in multiple IPUs. For instance, in the case of pipelining with two IPUs, the first six layers are placed on the first IPU with the index being 0, while the remaining four layers are placed in the second IPU with the index being 1. The placement of layers is dictated by the number of parameters at each layer. The number of parameters gradually increases from the first layer and peaks at the eighth layer before drastically decreasing with the last two layers. To further confirm our partitioning, PopVision is used to check the workload of each stage. Results from PopVision are in agreement with our pipeline configurations to achieve peak performance for IPUs.

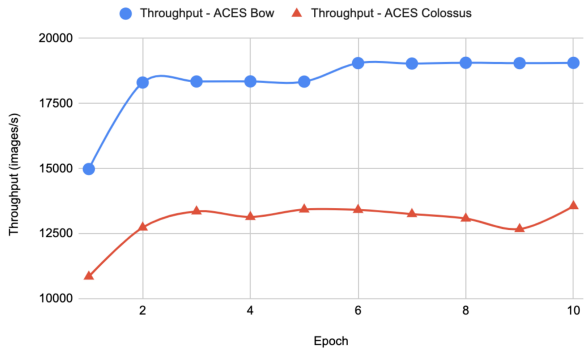

Figure 2: Image throughput variation of PyTorch ResNet50 model across 8 IPUs with epochs.

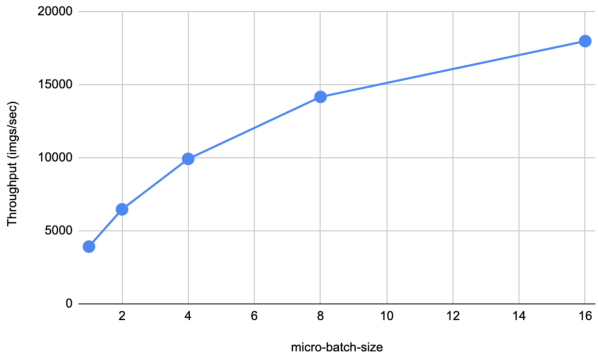

Figure 3: Image throughput variation of PyTorch ResNet50 model across 8 IPUs with epochs.

2.4 Performing the Calculation

Benchmarking on GPUs is performed on NVIDIA A100 (40GB, PCIe) [13] on the NSF FASTER cluster at Texas A&M University. These GPUs are composed over the PCIe fabric to dual-socket nodes powered by 64-core processors (two 32-core Intel Xeon 8352Y Ice Lake processors)

On the other hand, IPU calculations are performed on Graphcore Bow-IPU compute nodes provided by ACES. Each compute node includes 16 IPUs. Graphcore provides their own implementation of both Tensorflow and PyTorch through the Poplar software development toolkit. Graphcore also provides benchmarking tools and documentation for widely-used AI/ML models, and comprehensive guidance on ML workload environments [3, 6]. It is imperative to emphasize that benchmarking measurements for training are conducted starting from the second epoch onward. This is because ML models undergo compilation during the initial epoch, a procedure that usually demands a significant amount of time. Consequently,

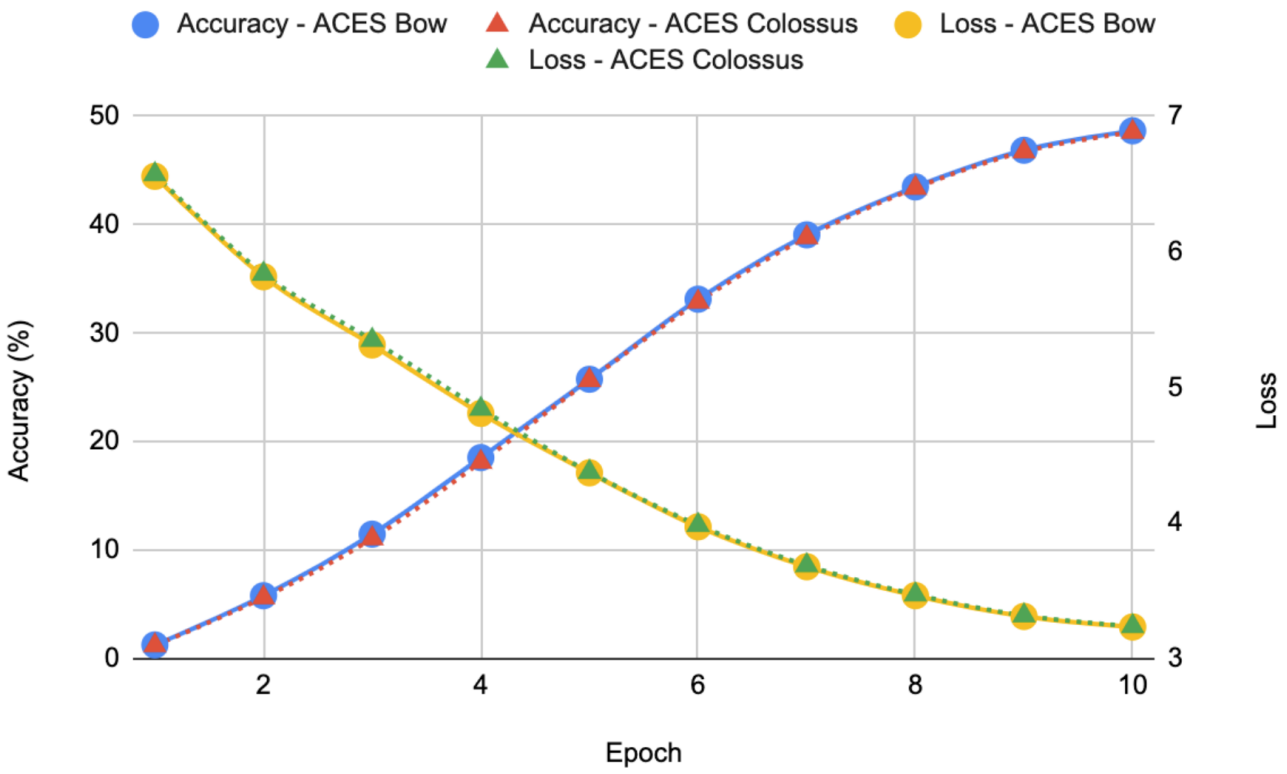

Figure 4: Image throughput variation of PyTorch ResNet50 model across 8 IPUs with epochs.

it is more precise to assess benchmarking criteria from the second epoch onward.

Graphcore IPUs allow training and inference to be performed in parallel with single or multiple hosts. The number of hosts (number of instances) can be specified using the command 'poprun' to launch different IPU nodes. Furthermore, the number of replicas (model copies) can also be defined to create multiple copies of the same graph of a model. Based on these two parameters, 'poprun' automatically allocates the correct number of IPUs for the given task. The total number of IPUs is a result of the multiplication between the number of instances and the number of replicas. Each host (instance) manages the input/output of data, but it does not directly affect the computation of IPUs. As a result, this parameter is specified based on the workload of data transferring between IPUs and hosts. In general, the number of instances is equal to or divisible by the number of replicas.

3 RESULTS AND DISCUSSION

3.1 Computer Vision model - PyTorch ResNet50

For benchmarking on IPUs with computer vision models, we used the same PyTorch ResNet50 model that was used in the previous paper [11, 12]. ResNet model [5] is a popular CNN model for its innovation of skip connections to address the gradient degradation problem. It also serves as the backbone for many models of image classification, object detection, semantic and instance segmentation, etc.

Figure 2 shows the image throughput of PyTorch ResNet50 varies with the number of epochs on ACES Bow and Colossus IPU systems respectively across 8 IPUs. Initially, for both Bow and Colossus IPU systems, the first epoch exhibits lower performance primarily because of graph compilation overhead. However, the performance improves and stabilizes after the second epoch. Therefore, the performance values reported in this study are obtained after 2 epochs. Also, we can see the performance comparison of the two systems is evident. On average, the Bow IPU system outperforms the Colossus system by approximately 42%. This is consistent with the comparison of the IPU frequencies. The Bow IPU frequency is 1.85 GHz. The Colossus IPU frequency is 1.325 GHz. And the difference is around 40%.

In deep learning models, batch size can impact how efficiently hardware resources are utilized and different batch sizes may lead to different levels of memory usage and computational efficiency. We study the image throughput variation of the PyTorch ResNet50 model with the micro-batch sizes across 8 IPUs (Figure 3). As shown in the figure, there is a clear positive correlation between the micro-batch size and the throughput. At smaller micro-batch sizes, the performance of throughput increases, indicating higher efficiency in processing images at a higher rate with micro-batch size. However,

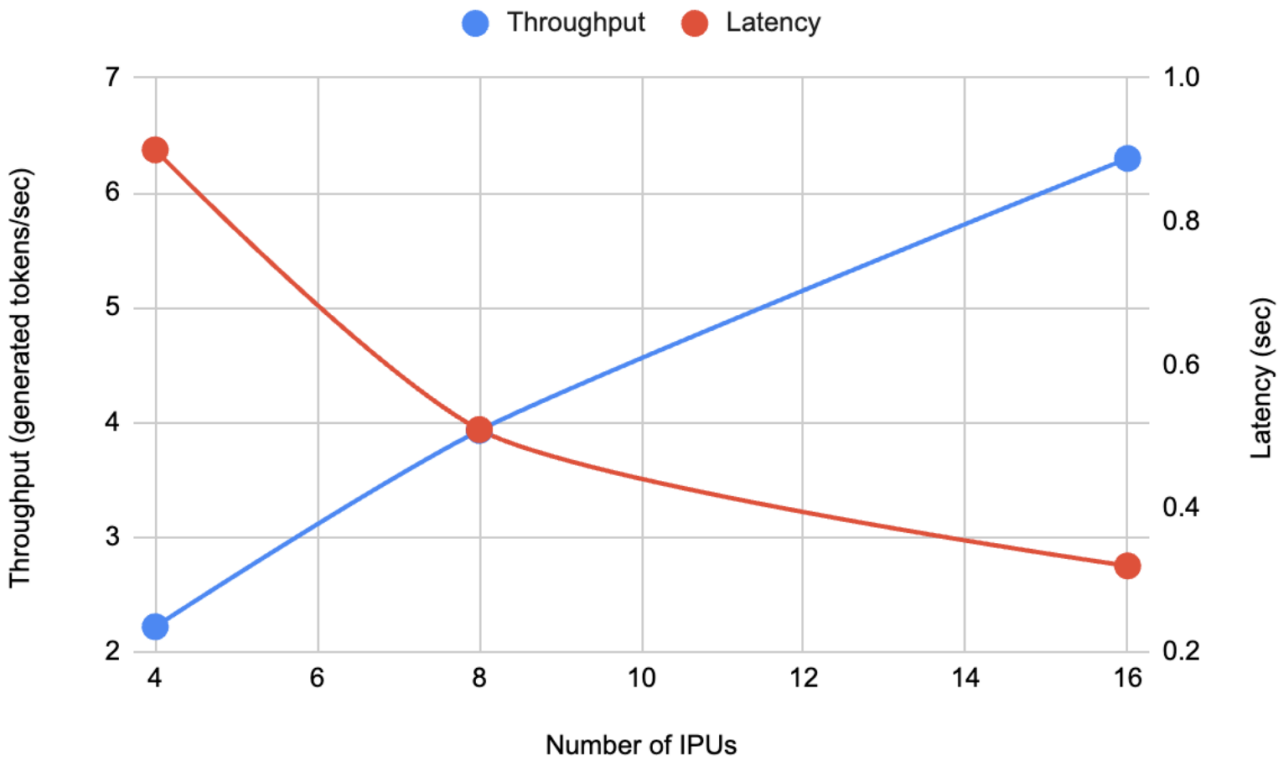

Figure 5: Sequence throughput and latency variation of LLaMa2-7B model inference with the number of IPUs.

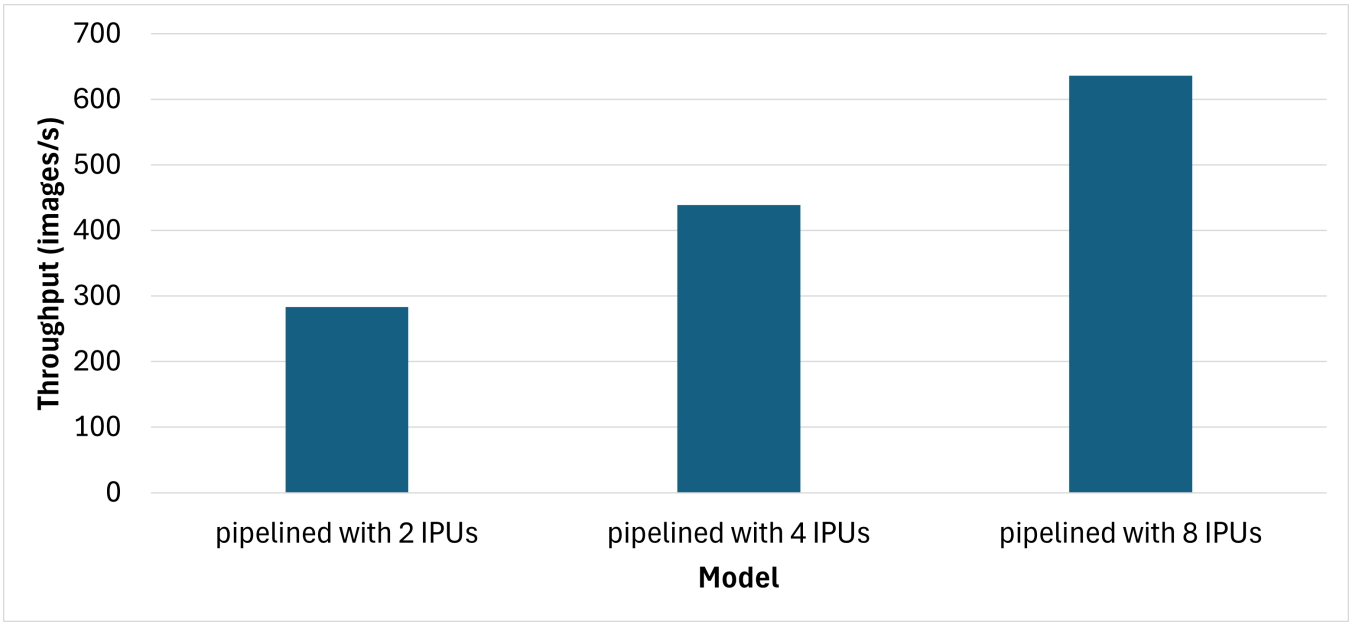

Figure 6: Performance of different pipelined models without replicason on Bow IPUs

at larger micro-batch sizes, the increase slows down because of the memory and computation being almost saturated.

Monitoring the training loss and accuracy of deep learning models over epochs can help assess whether hte models are converging.

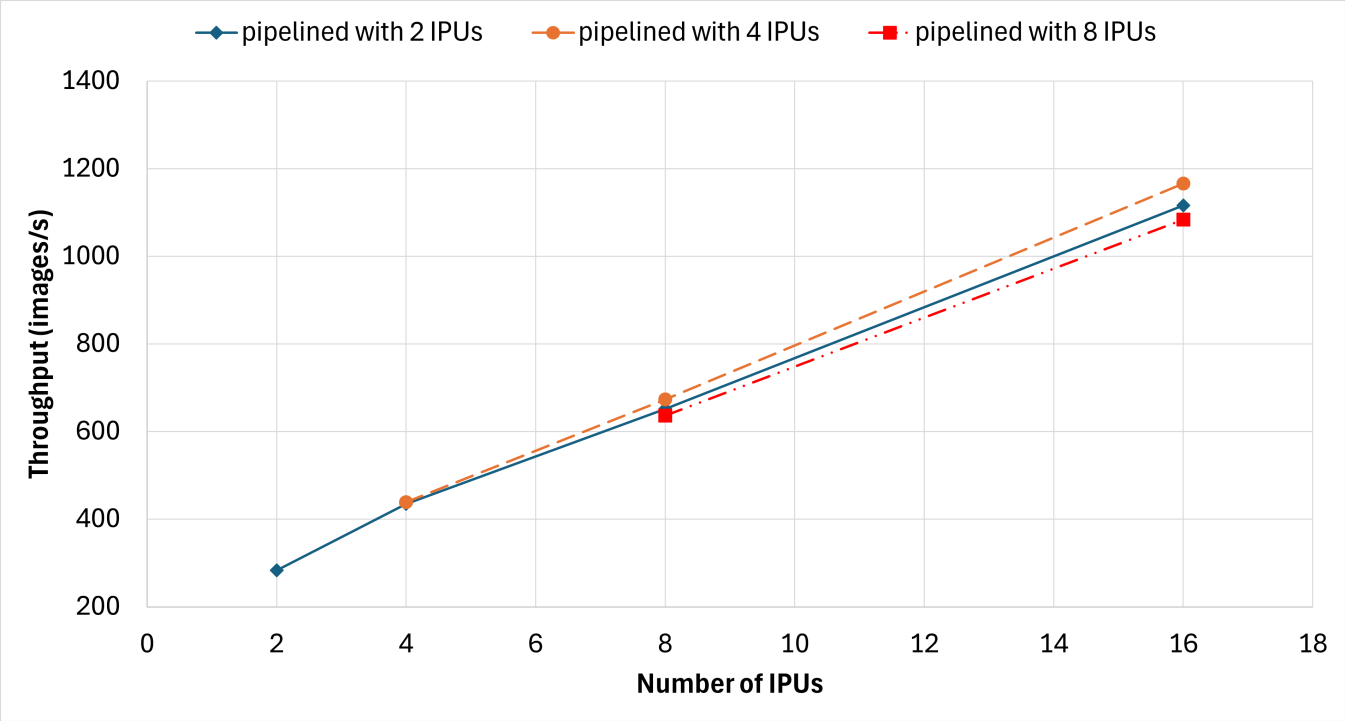

Figure 7: Performance of different pipelined models without replicas on Bow IPUs

Table 2: Performance of the 2-IPU pipelined model and GPU implementation

Hardware	Device Quantity	Micro BSize	Global BSize	Throughput (images/s)
NVIDIA A100	1	256	256	304.29
NVIDIA A100	2	256	512	567.32
NVIDIA A100	4	256	1024	1,077.34
Bow-IPU	2	2	80	289.04
Bow-IPU	4	2	160	435.30
Bow-IPU	8	2	320	720.17
Bow-IPU	16	2	640	1,116.62

Notes: BSize is batch size

Figure 4 depicts the training loss and accuracy of the PyTorch ResNet50 model with the number of epochs on both ACES Bow and Colossus IPU systems. We can see that the training loss and accuracy on the two systems are very close. The accuracy gradually increases from approximately 1.2% to nearly 50%, while the training loss steadily decreases from around 6.5 to about 3.2 over 10 epochs. In the final epochs, the trends start to plateau.

3.2 Large Language Model - LLaMa2-7B Inference

Figure 5 shows the performance of Meta's Large Language Model for Text Generation (LLaMa2-7B) with 7 billion parameters on the ACES Bow IPU system. This evaluation was conducted with micro-batch size of 2 and a sequence length of 1024. The model is obtained from the Gradient Hugging Face GitHub repository. From the figure, we can see that as the number of IPUs increases, the throughput (measured in generated tokens/sec) increases almost linearly, indicating enhanced processing efficiency when more IPUs are involved, and the latency decreases, indicating a quicker response. This proves the scalability and efficiency improvements achievable by allocating more IPUs to LLaMa2-7B model inference.

3.3 Conductivity Model

3.3.1 Performance of Pipelined Model on Bow-IPU. When the model is pipelined across various IPUs, there is an enhancement in performance, as illustrated in Figure 6. All model executions are conducted

on a suboptimal configuration with 20 iterations and 40 gradient accumulations. The mini-batch size is configured to the maximum for 2, 4, and 8-pipelined models, corresponding to 2, 4, and 8 images per mini-batch, respectively. However, the throughput does not scale linearly. One potential explanation for this phenomenon is the data exchange between different IPUs within an IPU-POD. Computing units experience delays while transferring data between IPUs, leading to a slight decline in performance.

3.3.2 Scaling with replicas. As depicted in Figure 7, different pipelined models achieve similar throughput when the number of IPUs being used is the same. For instance, with a total of 8 IPUs being used, the 2-IPU pipelined models with 4 replicas yield comparable outcomes to the 4-IPU pipelined model with 2 replicas and the 8-IPU pipelined model without any replicas. This indicates that both replicas and pipelining enhance performance almost identically.

However, effective pipelining, aimed at distributing the workload evenly across IPUs, has the potential to yield higher performance, as evidenced by the case of utilizing 16 IPUs (Figure 7). Using the same number of IPUs, the pipelined model where workload is distributed among all 4 IPUs outperforms the other two models in terms of throughput. This improvement stems from the flexibility provided by the architecture of the specialized model, allowing for nearly equal partitioning of all layers into pipelined blocks based on workload. In contrast, the 2-IPU pipelined model can only divide the model into two parts, thus limiting partitioning options given the model architecture. Conversely, the 8-IPU pipelined model, while employing an excess number of IPUs for a model consisting of 10 main blocks, results in inefficient utilization of IPU computing resources.

3.3.3 GPU and IPU comparison. Table 2 presents the performance results from training a specialized model on both the NVIDIA A100 GPU and the Bow-IPU. Our analysis reveals that GPUs require a sizable batch size to effectively train the model while maintaining high throughput. In contrast, although the Bow-IPU supports a minimal number of micro batch sizes, it still demonstrates significant performance gains. Furthermore, the global batch size for the IPU typically tends to be large due to the incorporation of gradient accumulation and model replication techniques. This arises from the fact that IPUs distribute workloads across tiles, whereas the NVIDIA A100 possesses significantly greater memory, enabling it to handle considerably larger batch sizes. More importantly, the performance of both platforms in the training process remains comparable across hardware platforms.

3.3.4 Limitations and Challenges. Despite the extensive application of optimization techniques to the conductivity model, the code in the IPU has not been optimized to achieve the maximum possible performance. There are several other optimization options available to enhance the model's performance, such as optimizing matrix multiplication operations with different data types and adjusting caching behaviors. These techniques can be tailored to specific tasks, presenting an opportunity for improvement in future implementations.

Porting code from GPU to IPU poses several challenges. Firstly, Graphcore gathers the majority of optimization options into a central variable, requiring careful review of the documentation to find equivalent options in PyTorch/TensorFlow for GPU. Moreover, effective pipelining requires a thorough understanding of IPU architecture, as the workload must be evenly distributed across all IPUs to achieve optimal scaling performance.

4 CONCLUSION

The training results of the conductivity model on Bow-IPU hardware demonstrate that IPU serves as a promising alternative platform for conducting ML tasks within the field of materials science and chemistry. Moreover, Graphcore offers various methods for efficiently transitioning a model from GPU to IPU. Among these techniques, pipelining stands out as a crucial approach for successfully migrating medium-to-large-scale ML models. Pipelining becomes particularly advisable when dealing with models that surpass the capacity of a single IPU, as it facilitates the partitioning of the model across multiple IPUs. Moreover, pipelined models, when using an equivalent number of IPUs, demonstrate comparable throughput during the training process to models trained with numerous replicas. Consequently, pipelining can serve as an alternative strategy for model replications across distributed systems. On the other hand, this study also delves into the performance and applicability of IPU. The IPU platform not only exhibits substantial performance improvements over its predecessor, the Colossus IPU, but also achieves training throughput comparable to that of the NVIDIA A100.

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); and staff at Texas A&M High Performance Research Computing.

REFERENCES

[1] Nathan Baker. 2024. *Unlocking a new era for scientific discovery with AI: How Microsoft's AI screened over 32 million candidates to find a better battery.* Retrieved March, 2024 from https://cloudblogs.microsoft.com/quantum/2024/01/09/unlocking-a-new-era-for-scientific-discovery-with-ai-how-microsofts-ai-screened-over-32-million-candidates-to-find-a-better-battery/

[2] 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, 141–151.

[3] Graphcore documents. 2024. *Graphcore documents.* Retrieved March, 2024 from https://docs.graphcore.ai/en/latest/

[4] Koji Fujimura, Atsuto Seko, Yukinori Koyama, Akihide Kuwabara, Ippei Kishida, Kazuki Shitara, Craig AJ Fisher, Hiroki Moriwake, and Isao Tanaka. 2013. Accelerated materials design of lithium superionic conductors based on first-principles calculations and machine learning algorithms. *Advanced Energy Materials* 3, 8 (2013), 980–985.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[6] TAMU HPRC. 2024. *TAMU HPRC Wiki.* Retrieved March, 2024 from https://hprc.tamu.edu/wiki/

[7] Jürgen Janek and Wolfgang G Zeier. 2023. Challenges in speeding up solid-state battery development. *Nature Energy* 8, 3 (2023), 230–240.

[8] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. 2018. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826* (2018).

[9] Mai Le, Hieu Le, Lihong Zhao, Xuqing Wu, Jiefu Chen, and Yan Yao. 2024. *Predicting Ionic Conductivity of Solid-State Battery Cathodes Using Machine Learning.* Retrieved March, 2024 from https://www.usnc-ursi-archive.org/nrsm/2024/papers/1316.pdf

[10] Mai Le, Alan Yao, Amie Zhang, Hieu Le, Zhaoyang Chen, Xuqing Wu, Lihong Zhao, and Jiefu Chen. 2024. *Predicting Ionic Conductivity of Solid-State Battery Cathodes Using Machine Learning, in preparation.*

[11] Abhinand Nasari, Hieu Le, Richard Lawrence, Zhenhua He, Xin Yang, Mario Krell, Alex Tsyplikhin, Mahidhar Tatineni, Tim Cockerill, Lisa Perez, et al. 2022. Benchmarking the performance of accelerators on national cyberinfrastructure resources for artificial intelligence/machine learning workloads. In *Practice and Experience in Advanced Research Computing*. 1–9.

[12] Abhinand Nasari, Lujun Zhai, Zhenhua He, Hieu Le, Suxia Cui, Dhruva Chakravorty, Jian Tao, and Honggao Liu. 2023. Porting AI/ML Models to Intelligence Processing Units (IPUs). In *Practice and Experience in Advanced Research Computing*. 231–236.

[13] NVIDIA. 2020. *A100 40GB PCIe Product Brief.* Retrieved March, 2024 from https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf

[14] Karl Steinbuch and Uwe AW Piske. 1963. Learning matrices and their applications. *IEEE Transactions on Electronic Computers* 6 (1963), 846–862.

[15] Dave Steinkraus, Ian Buck, and Patrice Y Simard. 2005. Using GPUs for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 1115–1120.