2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)

A Comprehensive Evaluation of Novel AI Accelerators for Deep Learning Workloads

Murali Emani* memani@anl.gov
Zhen Xie* zhen.xie@anl.gov
Siddhisanket Raskar* sraskar@anl.gov
Varuni Sastry* vsastry@anl.gov
William Arnold* arnoldw@anl.gov
Bruce Wilson* wilsonb@anl.gov

Rajeev Thakur* thakur@anl.gov
Venkatram Vishwanath* venkat@anl.gov
Zhengchun Liu* zhengchun.liu@anl.gov
Michael E. Papka*‖ papka@anl.gov
Cindy Orozco Bohorquez† cindy@cerebras.net

Rick Weisner‡ rick.weisner@sambanova.ai
Karen Li‡ xiaoyan.li@sambanova.ai
Yongning Sheng‡ yongning.sheng@sambanova.ai
Yun Du‡ yun.du@sambanova.ai

Jian Zhang‡ jian.zhang@sambanova.ai
Alexander Tsyplikhin§ alext@graphcore.ai
Gurdaman Khaira§ damank@graphcore.ai
Jeremy Fowers¶ jfowers@groq.com
Ramakrishnan Sivakumar¶ rsivakumar@groq.com

Victoria Godsoe¶ vgodsoe@groq.com
Adrian Macias¶ am@groq.com
Chetan Tekur¶ ctekur@groq.com
Matthew Boyd¶ matt@groq.com

*Argonne National Laboratory, Lemont, IL 60439, USA, †Cerebras Systems, Sunnyvale, CA 95085, USA, ‡SambaNova Systems Inc., Palo Alto, CA 94303, USA, §Graphcore Inc., Palo Alto, CA 94301, USA, ¶Groq Inc., Mountain View, CA 94041, USA, ‖University of Illinois, Chicago, IL 60637, USA

***Abstract*—Scientific applications are increasingly adopting Artificial Intelligence (AI) techniques to advance science. High-performance computing centers are evaluating emerging novel hardware accelerators to efficiently run AI-driven science applications. With a wide diversity in the hardware architectures and software stacks of these systems, it is challenging to understand how these accelerators perform. The state-of-the-art in the evaluation of deep learning workloads primarily focuses on CPUs and GPUs. In this paper, we present an overview of dataflow-based novel AI accelerators from SambaNova, Cerebras, Graphcore, and Groq. We present a first-of-a-kind evaluation of these accelerators with diverse workloads, such as Deep Learning (DL) primitives, benchmark models, and scientific machine learning applications. We also evaluate the performance of collective communication, which is key for distributed DL implementation, along with a study of scaling efficiency. We then discuss key insights, challenges, and opportunities in integrating these novel AI accelerators in supercomputing systems.**

***Index Terms*—Scientific Machine Learning, Deep Learning, Accelerators, Performance Evaluation, Benchmarking**

I. Introduction

Artificial Intelligence (AI) is emerging as a crucial component in several science domains, such as biology, high energy physics, and clean energy, to accelerate new scientific discoveries from data obtained from experiments and/or simulations. An AI-driven science application involves either a science experiment at an experimental facility or simulations carried out on supercomputers, coupled with a learning component that uses AI models to steer simulations, replace them either in part or entirely with surrogate models, solve a scientific problem as an *AI for science* workload, or a combination of the above. There will be a surge in scientific applications that require infrastructure to enable in-place data analysis at experimental facilities and AI capabilities integrated with large-scale models. The US Department of Energy (DOE) AI for Science Report [1], put forth by stakeholders from DOE labs, academia, and industry, cohesively highlights the need for tighter integration of the AI infrastructure ecosystem with experimental and leadership computing facilities. There is great emphasis on efficiently implementing Deep Learning (DL) models and exploiting novel architectures, especially reduced-precision AI accelerators. The DOE Advanced Scientific Computing Research (ASCR) report on extreme heterogeneity [2] lists challenges in integrating a broad spectrum of diverse hardware resources for science.

Recent advances in hardware, including heterogeneous systems and AI accelerators, will help researchers to advance the state of the art in scientific applications on powerful exascale supercomputers such as Aurora [3], El Capitan [4], and Frontier [5]. At the same time, we are witnessing the emergence of specialized AI accelerators, such as from Cerebras [6], SambaNova [7], Groq [8], and Graphcore [9], promising orders-of-magnitude improvement for various AI workloads and are being deployed at various HPC facilities such as AI Testbed at ALCF [10]. There have been increasing use cases in successfully porting and evaluating scientific Machine Learning (ML) applications on these accelerators [11]–[20]. AI accelerators are also expanding their capabilities to cater to a broader range of applications. For example, Cerebras created a software development kit to run computational fluid dynamics

978-1-6654-5185-7/22/$31.00 ©2022 IEEE
DOI 10.1109/PMBS56514.2022.00007

2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS) | 978-1-6654-5185-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/PMBS56514.2022.00007

applications with no AI models on the CS-2 system [21]. Graphcore Intelligence Processing Unit (IPU) systems have been explored to run stencil computations on structured grid problems [17]. The trend of enhancing their capabilities to accelerate traditional high-performance computing applications will continue to evolve rapidly. AI accelerators have already proven to have a tremendous impact on various scientific applications.

With the growing importance of AI accelerators in HPC facilities, it is critical to understand how these systems perform for different DL tasks. However, only a limited amount of work exists on evaluating these accelerators, predominantly with GPUs. There is no existing work on a comprehensive evaluation of various accelerators. In this paper, we perform a comprehensive evaluation of several AI accelerators with a mix of workloads, including core DL kernels, benchmark models, as well as scientific applications. *To the best of our knowledge, this work is the first of its kind in evaluating modern AI accelerators across diverse workloads.*

The contributions of this paper include the following:

- We present a detailed comparative overview of the hardware and software characteristics of modern AI accelerators.
- We present a first-of-a-kind methodology and evaluation of the performance of the accelerators with a diverse set of important DL workloads.
- We explain the implementation challenges and optimizations.
- We provide insights from this timely and critical experimental analysis.

The remainder of the paper is organized as follows: Section II discusses the background on AI accelerators and benchmarking tools and practices for both ML and traditional HPC applications. Section III presents a detailed overview of various AI accelerators involved in this study. Next, Section IV describes the workloads used in this study and their evaluations. We present the results in Section V followed by our key insights from these experiments in Section VI, and conclude in Section VII.

II. Background

GPUs have played an important role in ending the AI winter as well as keeping up with the increased compute demand of state-of-the-art ML models. Computer architects are moving towards the design of domain-specific architectures and dataflow-based spatial accelerators [22], [23], which makes use of optimizations such as data reuse and efficient mapping to underlying spatial architecture. Performance comparison and evaluation of DL models are discussed in [24]–[26], but they are limited to CPUs, GPUs, and TPUs. Though the emerging AI accelerators build upon core dataflow principles, they vary in how they leverage them to accelerate various aspects of ML, making certain architectures more suitable than others for a particular type of workload. Hence, the task of characterizing the performance of the accelerators for various workloads is essential and timely.

It is crucial to benchmark systems to help understand the intricacies in the performance of scientific machine learning applications. There is an urgent need to understand fair and effective benchmarking of ML applications that are representative of real-world scientific use cases, unlike the TOP500 list [27] that ranks supercomputers worldwide and publishes their performance numbers (in FLOPS) with High-Performance Linpack (HPL). Several benchmarking efforts have previously aimed to characterize the performance of ML workloads, including Deep500 [28], HPCAI500 [29], and HPL-AI [30]. Other efforts aimed at analyzing model performance include DAWNBench [31], DeepBench [32], Fathom [33], ParaDNN [34], HPE DLBS [35], XSP [36], and Mahon [37], but they primarily aim at CPU and GPU-based systems. MLPerf™ [38] is a community-driven standard to benchmark ML workloads, focusing on end-to-end performance metrics. MLPerf HPC [39] is a benchmark suite of large-scale scientific ML training applications, CosmoFlow [40], DeepCAM [41] and OpenCatalyst [42], driven by the MLCommons™ Association.

It is vital to gain a deeper understanding of the performance characteristics of the low-level fundamental kernels underlying ML algorithms, which are responsible for the most computation time and energy requirements. Studying the most efficient mapping of these kernels for the novel dataflow-based spatial accelerators is an essential first step. For this purpose, a mapping study of fundamental kernels such as GEMMs, convolutions (Conv2D, Conv3D), fully connected (FC) layers, and activations (e.g., ReLU), in the context of architectural features of dataflow-based spatial accelerators, is needed. Until now, there have been studies on general matrix-matrix multiplications (GEMM) primarily on GPU-based systems [43]–[50]; however, there is a lack of comprehensive studies on a variety of dataflow-based accelerators. Additionally, the work in [51] evaluates spatial accelerator architectures for matrix-matrix multiplications. It proposes an analytical cost model to optimize runtime and energy for GEMM routines to generate dataflow mappings. Techniques to map dataflows to spatial accelerators [52] have been studied in a limited scope for traditional ML workloads using methods such as genetic algorithms [53] and reinforcement learning [54]. Other efforts [55]–[61] have proposed ways to scale DL models by partitioning across available hardware by model or tensor-parallelism, but are limited to GPUs and TPUs [62].

III. Overview of Evaluated AI Accelerators

We compare five AI accelerators as part of this work. Below we describe the details of a node for each evaluated accelerator. Table I provides an overview of various features of a single unit of a node for these systems and compares their hardware characteristics and software stack.

1) NVIDIA® A100 GPU: The NVIDIA DGX A100 has eight NVIDIA A100 Tensor Core GPUs interconnected with NVLink® & NVSwitch® technology. Each A100 GPU features 19.5 TFLOPS of FP32 and 312 TFLOPS of FP16/BF16 performance, 40 GB of HBM memory, and 1.6 TB/s of memory

TABLE I: Features of evaluated AI accelerators

Feature	Nvidia A100	SambaNova Cardinal SN10	Cerebras CS-2	Graphcore GC200 IPU	Groq GroqCard
Compute Units	6912 Cuda Cores, 432 Tensor Cores	640 PCUs	850,000 Cores	1472 Compute cores	Single large core with 409,600 MACCs
On-Chip Memory	192 KB L1 40 MB L2	$>$ 300 MB	40 GB	900 MB	230 MB per GroqChip
DRAM	40 GB HBM2 & 1.7 TB DRAM	12 TB DDR4 off-chip memory	N/A	1 TB streaming memory	1TB DDR4
Process	7nm	7nm	7nm	7nm	14nm
Software Stack Support	TensorFlow, PyTorch, ONNX, MxNET, CUDA	SambaFlow™, PyTorch, TensorFlow	TensorFlow, PyTorch, Cerebras SDK	TensorFlow, PyTorch, ONNX, PopArt	TensorFlow, PyTorch, ONNX
Precision support	TF32, FP32, FP16, BF16	FP32, BF16, Int32, Int16, Int8	FP32, FP16, cbfloat	FP32, FP16	FP32, FP16, Int8
Interconnect	NVLink	Infiniband-based	Ethernet-based	IPU Link	RealScale™

bandwidth. The DGX A100 has full connectivity between all eight GPUs, and the communication bandwidth between any two GPUs is up to 300 GB/s (600 GB/s bidirectional). We use A100 as the baseline for comparison with other accelerators.

NVIDIA A100 supports the common DL stack, including TensorFlow, PyTorch, and MxNet. A100 also provides a variety of libraries, such as cuDNN, cuBLAS, cuFFT, cuSPARSE, and NVIDIA Collective Communication Library (NCCL)®, which are based on the Compute Unified Device Architecture (CUDA) platform and optimized to provide state-of-the-art performance in computations and communication on GPUs.

2) SambaNova DataScale® SN10-8R System: We use a SambaNova DataScale SN10-8R rack system which consists of two DataScale SN10-8 nodes interconnected with an InfiniBand-based fabric. Each SN10-8 node has a host module (1.5TB and 128 cores) and eight SambaNova Systems Cardinal SN10™ Reconfigurable Dataflow Units™ (RDUs). Each SN10 RDU chip contains 640 Pattern Memory Units (PMUs) and 640 Pattern Compute Units (PCUs), and a total of 320 MB of on-chip memory among all PMUs with 150 TB/s on-chip memory bandwidth. With BF16, the peak performance of one Cardinal SN10 RDU chip is above 300 TFLOPS. SN10 RDU chips on a system are interconnected via the RDU-Connect™ to enable both model and data parallelism, where each link is 64 GB/s.

The SambaNova system features SambaFlow™, a complete software stack designed to advance developer productivity and is also fully integrated with standard frameworks, such as TensorFlow and PyTorch. SambaFlow enables the user to extract, compile, optimize, assemble, and execute the optimal dataflow graph of any model on this system.

3) Cerebras CS-2 system: At the heart of the Cerebras CS-2 system is the second-generation Wafer-Scale Engine (WSE-2), a massive parallel processor built from a single 300mm wafer. It offers 850,000 cores designed for sparse linear algebra, with peak performance of 75 PFLOPS with sparse FP16 and 7.5 PFLOPS with dense FP16, with 40 GB of onboard SRAM distributed across the WSE, ensuring that each core can access its local memory in a single clock cycle. The CS-2 system is a network-attached system [1]. Instead of connecting through a PCIe or similar protocol, the CS-2 system can act as a complete, independent compute node. This unusual capability enables system-level heterogeneity.

The Cerebras Software Platform supports popular machine learning frameworks, TensorFlow and PyTorch. The Cerebras SDK allows programmers to write lower-level code that targets the WSE's microarchitecture directly using a domain-specific programming language called the Cerebras Software Language (CSL). The sheer scale of the WSE enables models of very large size and makes it easier for one to program these models without having to reason with distributed memory programming. For models up to a billion parameters, the entire network can be held on-chip at once, with network layers mapped directly to regions of cores. Models of more than a billion parameters are executed using a "weight-streaming" mode, where activations are held on-chip and weights are streamed one layer at a time.

4) Graphcore IPU-M2000: The Graphcore IPU-M2000 is powered by four GC200 IPU (Intelligence Processing Unit) processors and delivers 1 PFLOPS of FP16 performance, with 3.6 GB on-chip memory and up to 256 GB of Streaming Memory. Each GC200 IPU has 1,472 processor cores, running 8832 independent parallel program threads with 250 TFLOPS of FP16 performance. Each GC200 IPU holds 918 MB on-chip memory with a bandwidth of 47.5 TB/s. The IPU gateway manages communication between IPUs via the IPU-Links with 320 GB/s communication bandwidth over 64 IPUs or 16 IPU-M2000s (Pod64). Scaling over multiple Pod64 machines is achieved by dual-port 100 Gb/s Ethernet NIC provides IPU over Fabric (IPUoF) connectivity.

Graphcore's Poplar SDK handles the compilation of an ML framework-level code, optimal data distribution over IPUs' memory, and execution of parallel threads on multiple IPU processors using the Bulk Synchronous Parallel (BSP) scheme. It uniquely uses phased execution, which exploits the fine-grained, MIMD nature of the IPU by ensuring there are no race conditions, livelocks, or deadlocks during parallel execution

[1]The Cerebras Wafer-Scale cluster that integrates multiple CS-2 systems in data parallel flow was not available at the time of evaluation

across the 1,472 processor cores and their tightly coupled memory, by compiling the timing of the compute, sync, and exchange phases into an executable graph. This makes the IPU well suited for common ML workloads and irregular workloads. Standard ML frameworks including TensorFlow, PyTorch, ONNX, and PaddlePaddle are fully supported along with access to PopLibs through Poplar C++ API.

5) GroqNode: The GroqNode server GN1-B8C consists of eight GroqCard accelerators. The Groq architecture was designed to deliver orders of magnitude lower latency versus legacy architectures. Key contributing features for doing so are the monolithic, single-core engine combined with spatial streaming dataflow. Groq uses its tensor streaming processor (TSP) architecture to enable it to store more models within memory and process data with high performance, which makes it ideal for making inferences from large datasets. Each GroqCard packages a single GroqChip processor into a standard PCIe Gen4 x16 and features 188 TFLOPS of FP16 and 750 TOPS of INT8 performance, 230 MB of on-chip memory, and 80 TB/s of on-chip memory bandwidth. The eight GroqCards are directly connected via the RealScale chip-to-chip network to enable near-linear multi-card, multi-server, and multi-rack scalability with no need for external switches.

The GroqWare software stack has two primary design entry points, either as PyTorch or TensorFlow programs or as a custom application on top of a bare-metal programming interface. Groq Compiler is used to deploy state-of-the-art deep learning Models that are trained in PyTorch, TensorFlow, and ONNX. Granular-level programming on GroqChip can be made by the Groq API. The GroqView™ profiler helps in improving the developer workflow.

IV. Evaluated Workloads

To evaluate the accelerators for a diverse set of AI models, we consider the workloads shown in Table IV. This includes (i) four fine-grained and key DL primitives, namely, GEMM, Conv2D, RNN, and ReLU, (ii) three standard benchmarks derived from MLPerf [38], U-Net, BERT-Large, and ResNet-50, (iii) two AI-driven scientific machine learning applications, X-ray Bragg diffraction peak detection (BraggNN [63]) and drug discovery for precision medicine (Uno [64]), and (iv) scaling benchmarks for collective communications and data-parallel training. Section IV provides additional details for this choice and associated metrics.

We chose this set of workloads because they are commonly used in the DL community, and all the evaluated systems support them. Other models exist that demonstrate the true potential of some of these systems. Still, evaluating such models is not the focus of this work as we aim to understand the performance across all the evaluated systems and not in isolation. We measured the performance of a model in terms of throughput in training mode for all evaluated accelerators except GroqNode and latency for the inference mode on GroqNode and A100s. Table V lists the software development environments used.

Performance Metrics: Depending on the type of workload mentioned in Table IV, suitable performance metric is chosen to provide useful insights. For DL primitives, *floating point operations per second* (FLOPS) is used as the performance metric, and the TFLOPS calculation method of each primitive is defined in Section IV. For benchmarks *throughput* - the processed data samples per second (e.g., images, sequences, pixels per second) is used as performance metric. For scientific applications, in addition to throughput, time to solution is also reported. For distributed experiments, we measure the bandwidth per device and the scaling efficiency. We use NVIDIA A100 as the baseline for our evaluations and compare and contrast the performance of AI accelerators. For the granularity of the compute, all DL primitives run on a single device for each system. For all other workloads, multiple devices are used and these configurations are discussed for each experiment in V. The software development environments used for each system are listed in the Table V.

A. Deep Learning Primitives

We define DL primitives as the basic building blocks of any deep learning model. The primitives we use are the most commonly used kernels underlying most DL workloads, namely, dense matrix multiplication, convolution layer, ReLU, and recurrent layer. These kernels are also chosen by the DeepBench [32] suite. Each primitive kernel is run with all supported precisions on every system. In particular, all batch sizes listed in this section are used for single precision (FP32/TF32). For a fair comparison to use the same memory size, we ran a batch size for half-precision (FP16/BF16) which is twice the batch size of single precision. Also, mixed precision in CS-2 runs is equivalent to FP16 in A100s.

1) General Matrix Multiply Layers: The General Matrix Multiply (GEMM) kernel is at the heart of most DL models. GEMMs are used to implement fully connected layers, part of convolution layers, and they are building blocks for recurrent layers. The performance in FLOPS for the GEMM operation for matrix sizes of (M, K) and (K, N) is calculated as: $FLOPS_{GEMM} = \frac{2*M*N*K}{t}$, where t is the measured execution time. The configurations used for evaluation of GEMM kernels are listed in Table VI chosen from the DeepBench suite [32], although these systems are capable of handling significantly larger matrix sizes. We also evaluated GEMMs with a sweep of matrix sizes up to the maximum value that can fit on the memory of a single device on each system.

2) Convolution Layers: Convolutions make up the vast majority of FLOPS in networks that operate on images and videos and form important parts of networks such as speech and natural language modeling, thus making them perhaps the single most important layer from a performance perspective. For the 2D Convolution operation (Conv2D) on input dimension of width w, height h, and channel c, FLOPS is calculated as: $FLOPS_{Conv2d} = \frac{2*W_o*H_o*k_f*c*f_w*f_h*b}{t}, W_o = \frac{w+2*Pad_w-f_w}{Stride_w}t, H_o = \frac{h+2*Pad_h-f_h}{Stride_h}t$, where k_f is the number of filters each of dimension (f_w, f_h) with padding (Pad_w, Pad_h) and stride ($Stride_w$, $Stride_h$), W_o and H_o are the

effective width and height after applying a filter, b is the batch size of input data, and t is the measured execution time of each step. Table VII lists the configurations used.

3) ReLU Layers: Rectified Linear Unit (ReLU) is the most commonly used activation function in DL models. ReLU takes one comparison and one multiplication for each input. The FLOPS of ReLU is calculated as: $FLOPS_{ReLU} = \frac{w*h*c*b}{t}$, where w, h, c, and b are the width, height, number of channels, and batch size of input data, and t is the measured execution time of each step. Table VIII lists the configurations used.

4) Recurrent Layers: Recurrent neural networks (RNN) are a class of neural networks that are powerful for modeling sequence data, such as time series or natural language. There are various types of recurrent cells, such as vanilla RNNs, LSTMs, and GRUs. The FLOPS of LSTM is calculated as: $FLOPS_{LSTM} = \frac{2*4*s*i*h*b+2*4*s*h*h*b}{t}$, where i and h represent the input and hidden dimension, and s and n represent the time step and batch size of input data. The configurations used in this work are listed in Table IX.

B. Benchmarks

In this section, we briefly describe the benchmarks and associated parameters used for our evaluation study.

1) U-Net: **U-Net** [65] is a convolutional neural network for biomedical image segmentation. The model used in this work aims at the task of brain MRI image segmentation [66]. The reference implementation is available at [67].

2) BERT: **BERT** (Bidirectional Encoder Representations from Transformers) [68] is a transformer model that is pre-trained on the English language corpus dataset in a self-supervised manner. It learns the inner representation of the text, which is used to extract features for downstream tasks. In this work, we used the pretraining phase of the BERT-large model with Wikipedia and BookCorpus datasets. For the inference mode evaluation, we used the DistilBERT model implementation.

3) ResNet-50: **ResNet-50** [69] is a deep residual network, which is a subclass of convolutional neural networks with 50 layers, popularly used for image classification tasks. In this work, we ran the model with the ImageNet dataset [70] and ran both training and inference with single and half floating point precision types.

C. Scientific Machine Learning Applications

Some science applications have been successfully ported and run on these AI accelerators. We chose two scientific machine-learning applications from different science domains that are important to DOE missions. These specific applications were chosen as they are ported on all the chosen systems, which helps to evaluate their performance across those accelerators.

1) BraggNN: X-ray diffraction-based microscopy techniques, such as High Energy X-ray Diffraction Microscopy (HEDM), rely on the high precision (sub-pixel) position of Bragg diffraction peaks. These positions are conventionally computed by fitting the observed intensities to a theoretical peak shape, such as pseudo-Voigt [63]. As experiments become more complex and detector technologies evolve, the computational cost of the conventional method becomes the main bottleneck of in-situ experiments. BraggNN [63] is a neural network designed to localize Bragg peak positions much more rapidly than conventional pseudo-Voigt peak fitting. An appropriately trained BraggNN can extract the Bragg peak information significantly faster than traditional analytical methods. Thus, a key challenge is providing trained BraggNN models at timescales compatible with the experiments to accommodate the extremely high data rates at modern synchrotron beamlines [11]. The reference PyTorch implementation is at [71].

2) CANDLE Uno: CANDLE [64] focuses on building a single scalable deep neural network that can address problems related to precision medicine for cancer cure. Here, we focus on one application, Uno, which aims to predict the drug response based on molecular features of tumor cells and drug descriptors. The reference implementation is hosted at [72]. Uno is evaluated with datasets of two sizes, a smaller dataset with 61876 training samples, and a larger dataset with about 20 million training samples. We use an *AUC* configuration to train the model till a certain validation loss value (0.0054) is met when we measure the model throughput.

D. Collective Communication and Scalability

We measured the network bandwidth incurred with collective communication calls, which constitute a significant portion of a distributed implementation. We also performed a scalability study to evaluate the performance of models with an increasing number of devices.

1) Collective Communication Bandwidth: Neural networks today are often trained across multiple devices. Synchronous techniques rely on keeping the parameters on all instances of the model synchronized, usually by making sure all instances of the model have the same copy of the gradients before taking an optimization step. The primitive usually used to perform this operation is called *All-Reduce*. This collective is observed to be the most dominant of the collective communication operations in a typical DL training run, hence we chose to evaluate the bandwidth with this operation. The configurations used in this experiment include 16,777,216 floats for 2, 4, and 8 ranks and 64,500,000 floats for 16 ranks.

2) Scalability Study: For this study, we performed an evaluation to understand how a model's throughput varies with an increasing number of devices. This exercise is important and timely, as these accelerators would need to scale out to address challenges with larger problem sizes in the near future. We scaled the number of devices of A100, SN10 RDU, and GC200 IPU from 1 to 8 running U-Net with the Kaggle-3m dataset on SN10-8 and ResNet-50 with CIFAR-100 dataset on A100 and IPU-M2000 systems. We carried out both strong and weak scale runs. For strong scaling, the cumulative batch size is fixed at 128 for all systems, e.g., batch size of 128 for 1 A100 GPU and batch size of 16 for 8 A100 GPUs. For weak scaling, the batch size on each device is fixed batch at

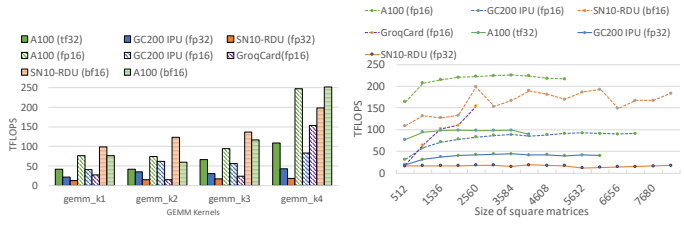
128, and the cumulative batch size increases as the number of devices grow.

V. Implementation and Results

Below we discuss the implementation details of our workload evaluations, their results, and observations. For baseline implementations on A100 GPUs, we plan to explore additional optimizations as part of our future work.

A. Evaluation of DL Primitives

We implemented the DL primitives with optimizations, such as pipelining, to exploit the advantages of the dataflow architecture underlying these accelerators. These optimizations help maximize data reuse by taking advantage of the larger capacity of SRAM compared to traditional architectures such as GPUs. For DL primitives experiments, this is achieved by repeating a layer of particular primitive multiple times, thereby enabling pipelined execution.[2]



(a) GEMM performance (b) GEMM scaling study

Fig. 1: GEMM Evaluation

1) General Matrix Multiply Layers: We ran GEMM kernels with two test cases: (i) a set of configurations from Table VI and (ii) a scaling study to stretch the matrix sizes until they saturate a device. Figure 1(a) shows the performance for the selected set of configurations and Figure 1(b) shows the scaling performance. For full-precision and half-precision modes (`TF32/FP32, FP16`), A100 reported highest FLOPS, while for `BF16` SN10-RDU has higher performance than A100 for first three kernels, while has lower performance for `gemm_k4` with larger matrix sizes. It can be observed that each system saturates at different matrix sizes given the limits on on-chip memories. Also, on each system, runs with lower precision could accommodate larger sizes than with higher precision because of lower memory requirements. We also observed a sub-linear speedup with increasing matrix sizes for all systems other than Groq, which has a 20-cycle loading time for the weights matrix to load into the MXM plane. Then it's linear results for a larger activation matrix.

The parameter `batch_size` is optimized for each system to take full advantage of available SRAM. Strategies such as on-device loops and repeating of GEMM layers are used to further improve performance for each system. For the SN10-RDU system, in addition to these general optimizations, the parameter `micro-batch_size` is also optimized to match the dimension of a matrix. The equation used to calculate FLOPS is modified to account for a change in computing caused by these optimizations. In addition to the model performance metrics, we also profiled and characterized the workloads on the evaluated accelerators. Specifically, we measured the compute and memory utilization on each device with kernel `gemm_k4` for GEMM primitive listed in Table X.[3] The compute utilization numbers show that compilers can do an effective job of mapping computation to their chips on each system.

2) Convolution Layers: Figure 2 shows the performance of the Conv2D primitive across different systems with supported precision. The list of kernel configurations used in this work exhibit varying behavior. Kernels `conv_k1_fw` and `conv_k2_fw` are memory-bound, whereas `conv_k3_fw` and `conv_k4_fw` are compute-bound. On A100, for the forward pass in training mode, half-precision cases have an average of $1.70\times$ performance improvement compared to single-precision cases due to different performance support for the two kinds of formats. A100 performs worse on `conv_k1_fw` and `conv_k2_fw` than `conv_k3_fw` and `conv_k4_fw`, which shows that A100 accelerates compute-intensive convolution operations better. The same observation also applies to SN10 RDU. GC200 IPU in half-precision mode exhibits larger speedups on an average $3.88\times$ over single-precision cases, indicating that GC200 IPU is more sensitive to the data format in the Conv2D kernel. For the backward pass in training mode, A100 performs best on `conv_k2_bw` and `conv_k3_bw`, SN10 RDU performs best on `conv_k1_bw` and `conv_k4_bw` kernels.

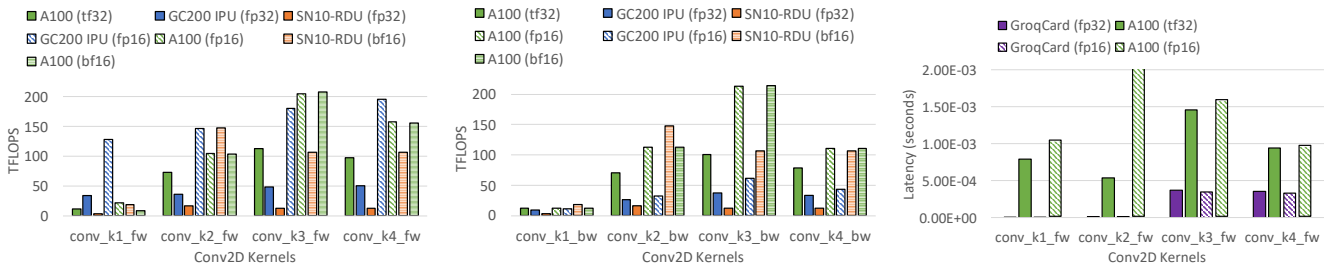
In inference mode, GroqCard has two orders of magnitude lower latency than A100 on `conv_k1_fw` and `conv_k2_fw`. GroqCard has $4.57\times$ and $2.87\times$ lower latency than A100 on `conv_k3_fw` and `conv_k4_fw`. Similar latency is observed on GroqCard using single-precision and half-precision formats. Since convolutions are matrix-matrix multiplications, the dedicated MXM for matrix multiplications and the VXM for bitwise multiplications, combined with the idea of dataflow pipelines avoid write-backs to memory and allow for optimized performance.

3) ReLU Layers: Figures 3(a) and 3(b) show the performance of the ReLU primitive across different systems in training and inference mode respectively. In training mode, for all devices, half-precision (FP16 and BF16) cases get almost twice the performance than single-precision (FP32 and TF32) cases, because half-precision cases have only half the amount of memory accesses. Especially, SN10 RDU achieves the best TFLOPS in all cases because its memory access unit, Pattern Memory Unit (PMU), is designed for high-performance on-chip data transfer. In inference mode, GroqCard has the lowest latency in all cases.

4) Recurrent Layers: As seen from Figure 4, for the forward pass in training mode, the performance of GC200 IPU in

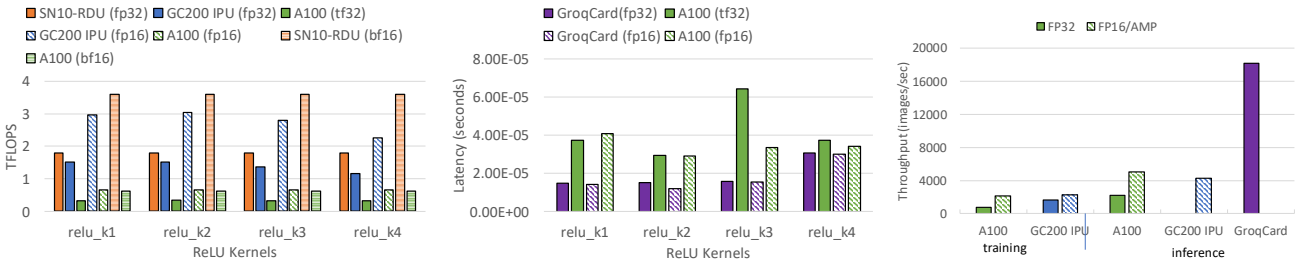
[2]Evaluation of CS-2 for DL primitives is not included as these kernels are not reflective of its potential performance.

[3]GEMM Size 1000 is used for GroqNode as this is the largest matrix size supported by GroqView at the time of experimentation with an older SDK.



(a) Training mode, forward pass (b) Training mode, backward pass (c) Inference mode, forward pass

Fig. 2: Conv2D Evaluation



(a) ReLU training mode (b) ReLU inference mode (c) ResNet-50

Fig. 3: ReLU and ResNet-50 Evaluation

single-precision cases is on average 1.93× higher than A100. GC200 IPU, SN10 RDU and A100 have similar half-precision performance. For the backward pass in training mode, GC200 IPU and A100 have almost the same performance in single-precision cases, whereas A100 performs much better than GC200 IPU in half-precision cases. SN10 RDU outperforms A100 and GC200 IPU in the backward pass and achieves an average of 1.57× and 2.53× performance improvement, respectively. In inference mode, GroqCard gains from batching as it uses matrix-vector multiplications that don't fill up the MXM planes. With increased batch sizes, the additional inputs fill the spare cycles in the schedule between bursts of weight loading onto the MXM plane which boosts performance.

B. Evaluation of DL Benchmarks

1) U-Net: Figure 5(a) shows a log-plot of the throughput (number of samples/s) of the U-Net model. The model was run with the largest image size that fits on the A100 i.e. 256x256, with a sweep of global batch sizes from 16 to 256. Here, we present results for two batch sizes, 32 and 256. The implementations on A100 and SN10-RDU use PyTorch, while IPU-M2000 uses TensorFlow, and CS-2 uses TensorFlow Estimator Framework[4]. For a smaller batch size of 32, throughput improvements observed for 8 SN10-RDUs, 1 CS-2, and 8 GC200 IPUs against 8 A100s are 2.1×, 4.9×, and 10× respectively. With a larger batch size of 256, these speedup factors are 0.7× [5], 3.1×, and 3.3× respectively. It is obvious that the speedup factors are low for large batch sizes with the increased memory consumption on each device. The interconnect bandwidth on the evaluated systems is sensitive to the batch size; distributed runs with batch size 256 vs. 32 result in a 3× difference for 8 A100s, 1.02× for 8 GC200 IPUs, and 0.93× for 8 SN10-RDUs.

When the number of devices are scaled from 1 to 8 across A100s, SN10 RDUs, and GC200 IPUs for a batch size of 32, the Scaling Efficiencies (SE) observed are 18.8%, 42%, and 79.6% respectively. Similarly, for a larger batch size of 256, the SEs measured are 52%, 28%, and 79.5% respectively. For IPUs, this is primarily due to efficient data loading from CPU to IPUs with multiple CPU instances per host

These AI accelerators also support U-Net with larger image sizes. On the CS-2, given the largest number of cores on a single device, U-Net can be trained on much larger image sizes as-is, while the runs on SN10-8 and GC-M2000 need data and model-parallel implementations.

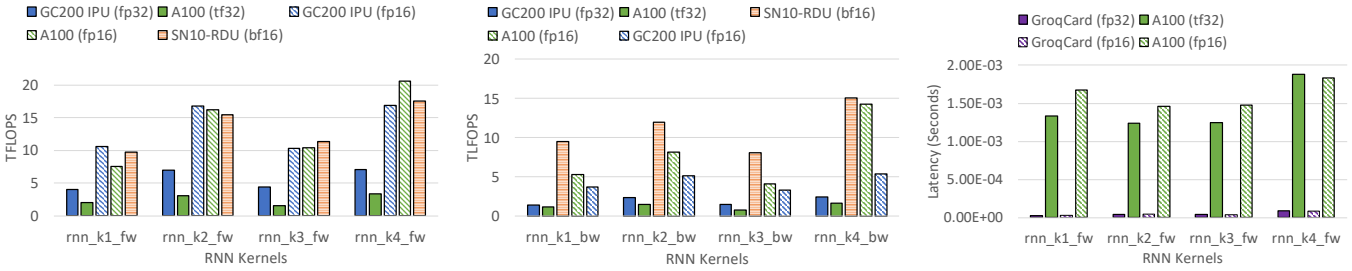
2) BERT: The throughput values of the BERT-large model on evaluated accelerators are shown in Figure 5(b). For training runs, we used BERT-Large with the Wikipedia Corpus data set. We tested a TF2 BERT implementation on A100, a TF Estimator BERT implementation on CS-2, and PyTorch BERT implementations on SN10-8 and IPU-M2000, with a global batch size of 256 and a maximum sequence length (MSL) of 128 for all runs.[6] We observe SEs of 97% and 93% on A100 and SN10-8 as we scale from a single device to 8 devices. Similarly, on IPU-M2000, we observe linear scaling to 8 IPUs. The performance of CS-2 is 2.3×[7] higher than that on 8 A100s, because of its ability to fully exploit

[4]The Cerebras U-Net implementation is in experimental face. A PyTorch implementation is work in progress.

[5]With the software release (v1.13) this speedup is increased to 2.1x

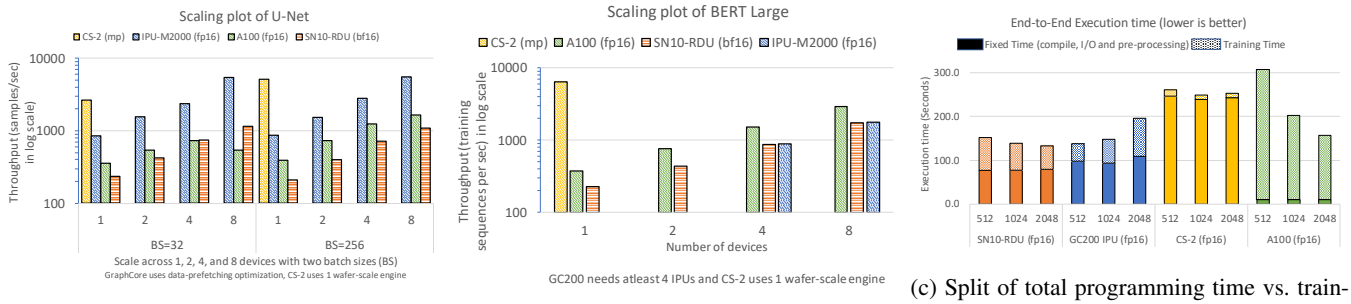
[6]Evaluation with a sweep of batch sizes and MSLs is work in progress.

[7]CS-2 performance increases 10% with a batch size of 1024



(a) Training mode, forward pass (b) Training mode, backward pass (c) Inference mode, forward pass

Fig. 4: RNN Evaluation



(a) U-Net (b) BERT (c) Split of total programming time vs. training time for BraggNN

Fig. 5: Performance evaluation of U-Net, BERT, and BraggNN

the compute, memory, and interconnect capability of the wafer-scale engine. Both SN10-8 and IPU-M2000 systems reported lower throughput numbers than A100. These values are 0.67× and 0.61× for 8 SN10 RDUs and 8 GC200 IPUs compared with 8 A100s. For inference mode runs on a single A100, GC200 IPU, and GroqCard with a BERT-Base model, DistilBERT [73], the throughput measured is 76, 658, and 1012 respectively. Compared to an A100, we observed 9× and 13× improvements for batch-1 inference on an GC200 IPU and GroqCard respectively. In GroqCards, the low latency of the SRAM loading at 10TB/s contributes to this speedup.

3) ResNet-50: The evaluation results for this model are shown in Figure 3(c) for both training and inference. The throughput values for half/mixed (FP16/amp) precision compared with full precision (TF32/FP32) for A100 and GC200 IPU systems are 2.75× and 1.25×, respectively. The inference-only Groq performs significantly better for this benchmark. The pipeline efficiently streams data from MEM to the MXM for the matrix multiplications and then to the VXM for performant ReLU. The ResNet-50 implementation for A100 uses PyTorch, whereas GC200 IPU uses Tensorflow. We tested for various combinations of batch sizes on all systems and have captured the best performance for comparison.[8]

C. Evaluation of Scientific Machine Learning Applications

1) BraggNN: For this application, we measure the end-to-end execution time in seconds and throughput for varying batch sizes and 500 epochs for 25464 samples. This application is run with FP16 precision on A100 and GC200 IPU, BF16 on CS-2, and mixed precision on SN10 RDU. CS-2 leverages its multi-replica mode of execution with 16 replicas on the wafer.[9] As observed from the throughput results in Table II, there are benefits of using a large batch size for GC200 IPU [10], but the throughput gets worse with a very big batch size because the training process changes from compute-bound to IO-bound. We further measure the split of the end-to-end execution time in Figure 5(c) which is composed of (a) model training time and (b) fixed time which includes fabric programming and data pipeline time. SN10 RDU and GC200 IPU achieve the lowest end-to-end execution time and achieve up to 1.55× and 1.46× speedup compared to A100. However, even though the fixed time on CS-2 is higher than other systems, it still reports the highest throughput. On A100, the fixed time is short, however, the model training time dominates the end-to-end execution time. For the SN10 RDU, the forward and backward graphs of BraggNN are mapped spatially on the same chip. With the inference mode, measuring the latency per image (microseconds), we observed 50 microseconds on GroqChip compared against 570 microseconds on an Intel Xeon CPU on average for an input size of 10K samples with batch size 1.[11]

2) CANDLE Uno: While the A100 and IPU-M2000 systems use the TensorFlow-Keras framework, SN10-8 and CS-2 use PyTorch and TensorFlow-Estimator-based implementation. For the compute-intensive AUC configuration of the Uno

[8]SN10-RDU and CS-2 do not currently support the ResNet-50 model.

[9]On A100, MIG mode was not a feasible option as it lacks inter-instance communication support. Evaluation with MPS mode is ongoing work.

[10]GC200 uses 2 IPUs, one for training and other for validation

[11]Experiments of BraggNN in inference mode on A100 is ongoing work.

TABLE II: BraggNN Throughput (in order of 1k samples/sec) with various batch sizes (BS)

System	BS=512	BS=1024	BS=2048
CS-2 (FP16)	1365.4	2463	2787.9
GC200 IPU (FP16)	478.0	350.6	219.9
SN10 RDU (BF16)	369.7	449.8	518
A100 (FP16)	53.9	65.5	73.7

TABLE III: Uno Performance Evaluation with Full Dataset

System	#Units	Batch size	Throughput (samples/sec)
CS-2 (mp)	1 CS2 WSE	2000	872258.7
GC200 IPU (FP16)	1 IPU	512	46123
SN10-8 (BF16)	2 RDUs	16	31958
A100 (TF32)	1 GPU	512	7567

model with a batch size of 32, a throughput of 10,184, 35,301, and 32,323 samples/s was observed on A100, SN10-8,[12] and GC200 IPU, respectively, i.e., a 3× speedup on the accelerators compared to the A100.[13] The measured throughput reflects the model's performance and does not include the data pipeline. Next, the Uno model was trained with a larger dataset to stress the I/O system and evaluate how it impacts the throughput. The results are listed in Table III.[14] SN10-8 uses a spatial mapping implementation that maps the model across 2 SN10 RDUs. The performance can be significantly enhanced by using a data-parallel framework to process the large number of data samples across multiple SN10 RDUs. On the CS-2 system, we make use of 9 worker nodes that efficiently handle the input data pipeline and data streaming for this large dataset. We can also deploy a multi-replica execution mode to achieve better chip utilization and throughput. Though we train with a batch size of 256 on a single GC200 IPU, we can accommodate larger batch sizes by using a data-parallel framework across IPUs.

From the results, we observe speedups of 115×, 6×, and 4.2× on CS-2, IPU-M2000, and SN10-8 respectively compared to A100. Given that the Uno application has a very large collection of data samples with a rather small model, it is inherently I/O-bound. This limitation can be overcome by using accelerators that have novel techniques to address the problem of I/O bottleneck with optimized data input pipelines.

D. Evaluation of Collective Communication and Scalability

1) Collective Communication Bandwidth: The `All_Reduce` bandwidth observed is shown in Figure 6(a) for the message sizes and ranks as mentioned in Section IV-D1. We test 2, 4, 8, and 16 devices on A100 and IPU-M2000 and 2, 4, and 8 devices on SN10-8 and GroqNode, all with FP32 precision. The observed bandwidths on A100 and IPU-M2000 are higher because they provide higher communication bandwidth of 300 GB/s and 320 GB/s using NVSwitch and IPU-Link, respectively. The bandwidth per device reduces as we increase the number of devices for A100 and GC 200 IPUs, whereas the SN10-RDU and Groq systems seem to saturate the achievable bandwidth as the number of devices increases. For the bandwidth utilization, A100 and IPU-M2000 achieve 33.5% and 15.4% on 16 devices, and SN10-8 and GroqNode reach 26.6% and 20% on 8 devices.

2) Scalability: Figures 6(b) and 6(c) show the achievable throughput scaling for U-Net in strong and weak scaling runs, respectively. We used an image size of 256 x 256 with a fixed batch size of 128 for all devices (fixed problem size) for strong scaling and a batch size of 128 (increasing problem size with scale) for weak scaling. Scaling is computed by the performance of an accelerator as we scale the number of devices over a single device of the same system in a data-parallel mode. A100 and IPU-M2000 exhibit similar scaling behavior, especially for strong scaling, which indicates that they can maintain the performance per device even though the batch size is gradually reduced. However, for SN10-8, the speedup of weak scaling is 24.1% better than that of strong scaling on average, which shows it is sensitive to batch size.

VI. Insights

Below we discuss key insights from these experimental evaluations and list some challenges that must be addressed for mainstream adoption of AI accelerators in scientific AI.

A. Porting efforts

As each AI accelerator has the software stack optimized and tuned for that platform, it is essential to port a deep learning model encoded in existing implementations, such as TensorFlow and PyTorch, to the respective accelerator software API, e.g., SambaNova SambaFlow, or preferred framework implementation, e.g., TensorFlow Estimator for TensorFlow code in Cerebras. Some accelerators can run models in native implementations (TensorFlow or PyTorch) as is; however, there are challenges in porting efforts due to a lack of support for some operators. Nonetheless, existing software stacks across these systems support many DL operators, and support for additional operations is expected to be available in future software releases.

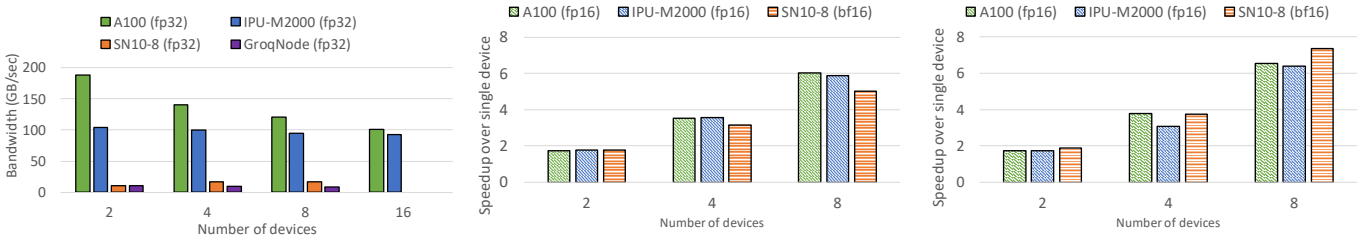
B. Impact of compute and memory capacity on throughput

On the SambaNova DataScale system, models with a large number of parameters can be served directly from the SN10 RDU on-chip SRAM given high bandwidth availability on RDUs in the range of a few 100s of TB/s. This also aids models on the Graphcore IPU-M2000 systems. Similarly, the CS-2 WSE with a large number of compute cores with on-chip memory is particularly well suited for running massive models efficiently. For smaller models, such as BraggNN, CS-2 offers a *multi replica* mode to run multiple copies of the same model using data parallelism, thereby increasing the chip utilization and net throughput. For models wherein the data processing or interconnect performance is critical, such as data-parallel training, we observe that the DGX A100 still

[12]Evaluation of Uno on 1 SN10 RDU is work in progress.

[13]Uno with AUC configuration was not run on CS-2 owing to smaller dataset size which does not use the wafer efficiently.

[14]Evaluation of Uno with same hyper-parameters is work in progress



(a) Bandwidth measurement (b) Strong scaling (c) Weak scaling

Fig. 6: Experimental results of bandwidth measurement and scaling studies.

shines due to the architecture of NVLink, NVSwitch, and the available bandwidth. For inference jobs, the architecture of the GroqChip enabled highly effective runs at a batch size of 1.

C. Compilation time

The common steps in a model execution involve the generation of the intermediate graph representation in the compilation phase, followed by optimization of the generated graph, and finally mapping of the graph onto the hardware units. The most critical component in a model execution is the compilation phase, which directly influences the mapping of the operators to leverage the hardware. It has been observed that the compile time can be significantly large on some of these systems. Minimizing the compile time would further help in the effective use of the accelerators.

D. Support for traditional HPC applications

A large number of science applications and simulations being run on supercomputers have no AI components in them. It is interesting to see how AI accelerators can push their capabilities to accelerate traditional computations as this area has gained traction recently. Though there is progress in the software stacks to run HPC simulation codes on these AI accelerators, it still needs to mature in order to cater to diverse HPC applications.

E. Determining the best accelerator for a given workload

Based on our experimental evaluations, it is evident that determining the most efficient accelerator for a given model is nontrivial. However, some observations can be made. Large compute-intensive models can run on the Cerebras CS-2 system without a distributed implementation. The SambaNova SN10-8 system can support models with large on-memory requirements and at scale. The performance advantage of spatial auto kernel fusion in a dataflow architecture vs non-spatial execution in a Von-Neumann architecture can often only be demonstrated with end-to-end applications (such as BraggNN and Uno), and not with individual kernels. The Graphcore IPU-M2000 system enabled highly effective distributed implementations, while GroqNode enabled efficient inference runs at a batch size of 1. However, given the variation in the hardware and software features, support for operators, supported precision, challenges with distributed implementation, and the scale of execution, it takes additional investigation to determine the best accelerator for a given model.

F. Challenges

Going forward, we foresee challenges with deeper integration of AI accelerators with large-scale computing facilities. For example, should AI accelerators be strongly coupled as part of a supercomputer's node design to minimize data transfer overheads, or should there be a dedicated AI accelerator partition or a cluster with high-bandwidth connectivity—a disaggregated system—wherein the simulation and learning components are partitioned across these resources? There is an urgent need to develop software and computing environments for seamless integration into computing facilities. For example, AI accelerators implement custom interconnect technologies that are different from the ones used in HPC centers. It is also critical to designing a mechanism to determine which AI accelerator would yield the best performance on an AI model, and how these models can be built and tuned for that particular hardware and software. Also essential is incorporating explicit science domain knowledge into AI systems and hardware to improve robustness and capabilities.

VII. Conclusions and Future Work

We presented an overview of different AI accelerators that are designed to boost the efficiency of machine learning tasks and are seeing increasing adoption in the ML community. We then presented a systematic study of these systems with a variety of workloads and provided detailed insights into the observed performance. The performance results indicate that the novel hardware and software features of the accelerators do enable efficient model execution. We also presented a list of challenges and opportunities based on the growing needs of the scientific machine learning community. As part of future work, we plan to continue the study with sparse tensor operations and additional scientific workloads, such as large language models (e.g., GPT), graph neural networks, AI-driven simulation applications, and novel implementations across platforms along with power efficiency metrics.

Acknowledgment

This work was supported by the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science User Facility, and by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. DOE under Contract No. DE-AC02-06CH11357.

APPENDIX

TABLE IV: Evaluated Workloads

Type	Tasks
DL Primitives	GEMM, Convolution, ReLU, RNN
Benchmarks	U-Net, BERT-Large, ResNet-50
Scientific Applications	Uno, BraggNN
Miscellaneous	Collective communications, Strong and weak scaling runs

TABLE V: Software Configurations

System	Software
Nvidia	CUDA 11.2, CuDNN 8, Tensorflow 2.6, PyTorch 1.12.1
SambaNova	SambaFlow 1.12
Cerebras	Cerebras Software 1.5
Graphcore	Poplar 2.4.0
Groq	GroqWare™ suite 0.8.5

TABLE VI: GEMM Configurations

Name	M	N	K
gemm_k1	64	1760	1760
gemm_k2	2560	64	2560
gemm_k3	1760	128	1760
gemm_k4	2560	2560	2560

TABLE VII: Convolution Configurations

Name	width (w)	height (h)	channel (c)	batch (b)	kernel	filter_w	filter_h	pad_w	pad_h
conv_k1	7	7	32	131072	32	3	3	0	0
conv_k2	14	14	128	4096	256	3	3	1	1
conv_k3	54	54	16	256	1024	3	3	1	1
conv_k4	128	128	32	1024	128	5	5	0	0

TABLE VIII: ReLU Configurations

Name	width (w)	height (h)	channel (c)	batch (b)
relu_k1	7	7	32	262144
relu_k2	14	14	128	16384
relu_k3	54	54	1024	128
relu_k4	128	128	128	128

TABLE IX: RNN Configurations

Name	time_step(s)	batch_size(n)	input_size(i)	hidden_size(h)
rnn_k1	50	64	256	256
rnn_k2	25	32	512	512
rnn_k3	25	16	512	512
rnn_k4	50	32	512	512

TABLE X: Profile metrics for `gemm_k4` kernel

System	Profiling Tool	Compute utilization	Memory utilization
A100	NSight	92.68%	76.15%
SN10-RDU	SambaFlow	84.38%	94.53%
GC200 IPU	PopVision	87.36%	83.14%
GroqCard	GroqView	76.62%	99.14%

REFERENCES

[1] R. Stevens, V. Taylor, J. Nichols, A. B. Maccabe, K. Yelick, and D. Brown, "AI for Science: Report on the Department of Energy (DOE) Town Halls on Artificial Intelligence (AI) for Science," 2020, doi: 10.2172/1604756. [Online]. Available: https://www.osti.gov/biblio/1604756

[2] J. S. Vetter et al., "Productive Computational Science in the Era of Extreme Heterogeneity. Report for DOE ASCR Basic Research Needs Workshop on Extreme Heterogeneity, 2019," 1 2018.

[3] "Aurora Supercomputing System," https://www.alcf.anl.gov/aurora.

[4] "El Capitan," https://www.llnl.gov/news/llnl-and-hpe-partner-amd-el-capitan-projected-worlds-fastest-supercomputer.

[5] "Frontier," https://www.olcf.ornl.gov/frontier/.

[6] "Cerebras CS-2 System," https://cerebras.net/product-system.

[7] S. Systems, "Accelerated computing with a reconfigurable dataflow architecture," https://sambanova.ai/, April 2021.

[8] "GroqChip," https://groq.com/products/.

[9] Graphcore, "Graphcore Intelligent Processing Unit," https://www.graphcore.ai/products/ipu, April 2021.

[10] "ALCF AI Testbed," https://www.alcf.anl.gov/alcf-ai-testbed.

[11] Z. Liu, A. Ali, P. Kenesei, A. Miceli, H. Sharma, N. Schwarz, D. Trujillo, H. Yoo, R. Coffee, N. Layad, J. Thayer, R. Herbst, C. Yoon, and I. Foster, "Bridging data center AI systems with edge computing for actionable information retrieval," in 3rd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP), 2021, pp. 15–23.

[12] A. Trifan, D. Gorgun, Z. Li, A. Brace, M. Zvyagin, H. Ma, A. Clyde, D. Clark, M. Salim, D. J. Hardy, T. Burnley, L. Huang, J. McCalpin, M. Emani, H. Yoo, J. Yin, A. Tsaris, V. Subbiah, T. Raza, J. Liu, N. Trebesch, G. Wells, V. Mysore, T. Gibbs, J. Phillips, S. C. Chennubhotla, I. Foster, R. Stevens, A. Anandkumar, V. Vishwanath, J. E. Stone, E. Tajkhorshid, S. A. Harris, and A. Ramanathan, "Intelligent Resolution: Integrating Cryo-EM with AI-driven Multi-resolution Simulations to Observe the SARS-CoV-2 Replication-Transcription Machinery in Action," bioRxiv 10.1101/2021.10.09.463779, 2021.

[13] M. Emani, V. Vishwanath, C. Adams, M. E. Papka, R. Stevens, L. Florescu, S. Jairath, W. Liu, T. Nama, and A. Sujeeth, "Accelerating scientific applications with SambaNova reconfigurable dataflow architecture," Computing in Science & Engineering, vol. 23, no. 02, pp. 114–119, 2021.

[14] A. Brace, M. Salim, V. Subbiah, H. Ma, M. Emani, A. Trifa, A. R. Clyde, C. Adams, T. Uram, H. Yoo, A. Hock, J. Liu, V. Vishwanath, and A. Ramanathan, Stream-AI-MD: Streaming AI-Driven Adaptive Molecular Simulations for Heterogeneous Computing Platforms. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3468267.3470578

[15] L. R. M. Mohan, A. Marshall, S. Maddrell-Mander, D. O'Hanlon, K. Petridis, J. Rademacker, V. Rege, and A. Titterton, "Studying the potential of Graphcore IPUs for applications in Particle Physics," 2020.

[16] S. Kulkarni, M. M. Krell, S. Nabarro, and C. A. Moritz, "Hardware-accelerated Simulation-based Inference of Stochastic Epidemiology Models for COVID-19," 2020.

[17] T. Louw and S. McIntosh-Smith, "Using the Graphcore IPU for Traditional HPC Applications," EasyChair Tech Report, 2021.

[18] Z. Zhang and S. Zohren, "Multi-Horizon Forecasting for Limit Order Books: Novel Deep Learning Approaches and Hardware Acceleration using Intelligent Processing Units," arXiv preprint arXiv:2105.10430, 2021.

[19] M. R. Wyatt, V. Yamamoto, Z. Tosi, I. Karlin, and B. Van Essen, "Is disaggregation possible for HPC cognitive simulation?" in 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC). IEEE, 2021, pp. 94–105.

[20] "Cerebras Wafer Scale Engine paired with Lassen Supercomputer," https://www.llnl.gov/news/llnl-pairs-worlds-largest-computer-chip-cerebras-lassen-advance-machine-learning-ai-research, August 2020.

[21] K. Rocki, D. Van Essendelft, I. Sharapov, R. Schreiber, M. Morrison, V. Kibardin, A. Portnoy, J. F. Dietiker, M. Syamlal, and M. James, "Fast stencil-code computation on a wafer-scale processor," in SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020, pp. 1–14.

[22] J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture," Commun. ACM, vol. 62, no. 2, pp. 48–60, Jan. 2019. [Online]. Available: http://doi.acm.org/10.1145/3282307

[23] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of Machine Learning Accelerators," 2020 IEEE High Performance Extreme Computing Conference, HPEC 2020, 8 2020. [Online]. Available: http://arxiv.org/abs/2009.00993http://dx.doi.org/10.1109/HPEC43674.2020.9286149

[24] J. Yin, A. Tsaris, S. Dash, R. Miller, F. Wang, and M. A. Shankar, "Comparative evaluation of deep learning workloads for leadership-class systems," BenchCouncil Transactions on Benchmarks, Standards and Evaluations, vol. 1, no. 1, p. 100005, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772485921000053

[25] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," 2019.

[26] E. Buber and B. Diri, "Performance analysis and CPU vs GPU comparison for deep learning," in 2018 6th International Conference on Control Engineering Information Technology (CEIT), 2018, pp. 1–6.

[27] "Top500 List: November 2020," https://www.top500.org/lists/2020/11/, 2021.

[28] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning," in 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019, pp. 66–77.

[29] Z. Jiang, L. Wang, X. Xiong, W. Gao, C. Luo, F. Tang, C. Lan, H. Li, and J. Zhan, "HPC AI500: The Methodology, Tools, Roofline Performance Models, and Metrics for Benchmarking HPC AI Systems," 2020.

[30] "HPL-AI Mixed-Precision Benchmark," https://icl.bitbucket.io/hpl-ai/, 2021.

[31] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Re, and M. Zaharia, "Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark," SIGOPS Oper. Syst. Rev., vol. 53, no. 1, p. 14–25, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3352020.3352024

[32] "DeepBench," https://github.com/baidu-research/DeepBench, 2021.

[33] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. Brooks, "Fathom: reference workloads for modern deep learning methods," in 2016 IEEE International Symposium on Workload Characterization (IISWC), 2016, pp. 1–10.

[34] Y. Wang, G. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU Platforms for Deep Learning," CoRR, vol. abs/1907.10701, 2019. [Online]. Available: http://arxiv.org/abs/1907.10701

[35] "HPE Deep Learning Benchmarking Suite," https://github.com/HewlettPackard/dlcookbook-dlbs/, 2021.

[36] C. Li, A. Dakkak, J. Xiong, W. Wei, L. Xu, and W. Hwu, "XSP: Across-Stack Profiling and Analysis of Machine Learning Models on GPUs," in 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 326–327.

[37] S. Mahon, S. Varrette, V. Plugaru, F. Pinel, and P. Bouvry, "Performance Analysis of Distributed and Scalable Deep Learning," in 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020, pp. 760–766.

[38] "MLPerf," https://mlperf.org/.

[39] "MLPerf HPC Benchmark Suite," https://github.com/mlcommons/hpc, 2021.

[40] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook, K. J. Maschhoff, J. Sewall, N. Kumar, S. Ho, M. F. Ringenburg, Prabhat, and V. W. Lee, "CosmoFlow: Using deep learning to learn the universe at scale," in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. IEEE / ACM, 2018, pp. 65:1–65:11.

[41] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, P. Prabhat, and M. Houston, "Exascale Deep Learning for Climate Analytics," in SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 649–660.

[42] L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, and et al., "Open Catalyst 2020 (OC20) Dataset and Community Challenges," ACS Catalysis, vol. 11, no. 10, p. 6059–6072, May 2021. [Online]. Available: http://dx.doi.org/10.1021/acscatal.0c04525

[43] H. Anzt, Y. M. Tsai, A. Abdelfattah, T. Cojean, and J. Dongarra, "Evaluating the performance of NVIDIA's A100 Ampere GPU for sparse and batched computations," in 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2020, pp. 26–38.

[44] D. Yan, W. Wang, and X. Chu, "Demystifying Tensor Cores to Optimize Half-Precision Matrix Multiply," in 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 634–643.

[45] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, "Performance, design, and autotuning of batched GEMM for GPUs," in International Conference on High Performance Computing. Springer, 2016, pp. 21–38.

[46] A. Abdelfattah, S. Tomov, and J. Dongarra, "Fast batched matrix multiplication for small sizes using half-precision arithmetic on GPUs," in 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2019, pp. 111–122.

[47] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers," in SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 603–613.

[48] K. Matsumoto, N. Nakasato, T. Sakai, H. Yahagi, and S. G. Sedukhin, "Multi-level optimization of matrix multiplication for GPU-equipped systems," Procedia Computer Science, vol. 4, pp. 342–351, 2011.

[49] D. Mukunoki, K. Ozaki, T. Ogita, and T. Imamura, "DGEMM using tensor cores, and its accurate and reproducible versions," in International Conference on High Performance Computing. Springer, 2020, pp. 230–248.

[50] S. Markidis, S. Chien, E. Laure, I. Peng, and J. S. Vetter, "NVIDIA Tensor Core Programmability, Performance & Precision," in 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Los Alamitos, CA, USA: IEEE Computer Society, May 2018, pp. 522–531. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPSW.2018.00091

[51] G. E. Moon, H. Kwon, G. Jeong, P. Chatarasi, S. Rajamanickam, and T. Krishna, "Evaluating Spatial Accelerator Architectures with Tiled Matrix-Matrix Multiplication," arXiv preprint arXiv:2106.10499, 2021.

[52] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," IEEE micro, vol. 40, no. 3, pp. 20–29, 2020.

[53] S.-C. Kao and T. Krishna, "GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm," in 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE, 2020, pp. 1–9.

[54] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning," in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020, pp. 622–636.

[55] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "AccPar: Tensor partitioning for heterogeneous deep learning accelerators," in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 342–355.

[56] J. H. Park, G. Yun, C. M. Yi, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism. USA: USENIX Association, 2020.

[57] A. Xu, Z. Huo, and H. Huang, "On the acceleration of deep learning model parallelism with staleness," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020.

[58] J. Geng, D. Li, and S. Wang, "ElasticPipe: An efficient and dynamic model-parallel solution to DNN training," in Proceedings of the 10th Workshop on Scientific Cloud Computing, ser. ScienceCloud '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 5–9. [Online]. Available: https://doi.org/10.1145/3322795.3331463

[59] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks." Proceedings of Machine Learning and Systems, vol. 1, pp. 1–13, 2019.

[60] A. Jain, A. A. Awan, A. M. Aljuhani, J. M. Hashmi, Q. G. Anthony, H. Subramoni, D. K. Panda, R. Machiraju, and A. Parwani, "GEMS: GPU-enabled memory-aware model-parallelism system for distributed DNN training," in SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–15.

[61] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using Megatron-LM," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476209

[62] "Google Tensor Processing Units (TPU)," https://cloud.google.com/tpu.

[63] Z. Liu, H. Sharma, J.-S. Park, P. Kenesei, A. Miceli, J. Almer, R. Kettimuthu, and I. Foster, "BraggNN: Fast X-Ray Bragg peak analysis using deep learning," IUCrJ, vol. 9, no. 1, 2022.

[64] "ECP CANDLE benchmarks," https://github.com/ECP-CANDLE/Benchmarks.

[65] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241.

[66] M. Buda, A. Saha, and M. A. Mazurowski, "Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm," Computers in biology and medicine, vol. 109, pp. 218–225, 2019.

[67] "U-Net implementation in PyTorch for FLAIR abnormality segmentation in Brain MRI," https://github.com/mateuszbuda/brain-segmentation-pytorch.

[68] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Commun. ACM, p. 84–90, may 2017.

[71] "BragNN," https://github.com/lzhengchun/BraggNN.

[72] "Uno: Predicting Tumor Dose Response across Multiple Data Sources," https://github.com/mateuszbuda/brain-segmentation-pytorch.

[73] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019.