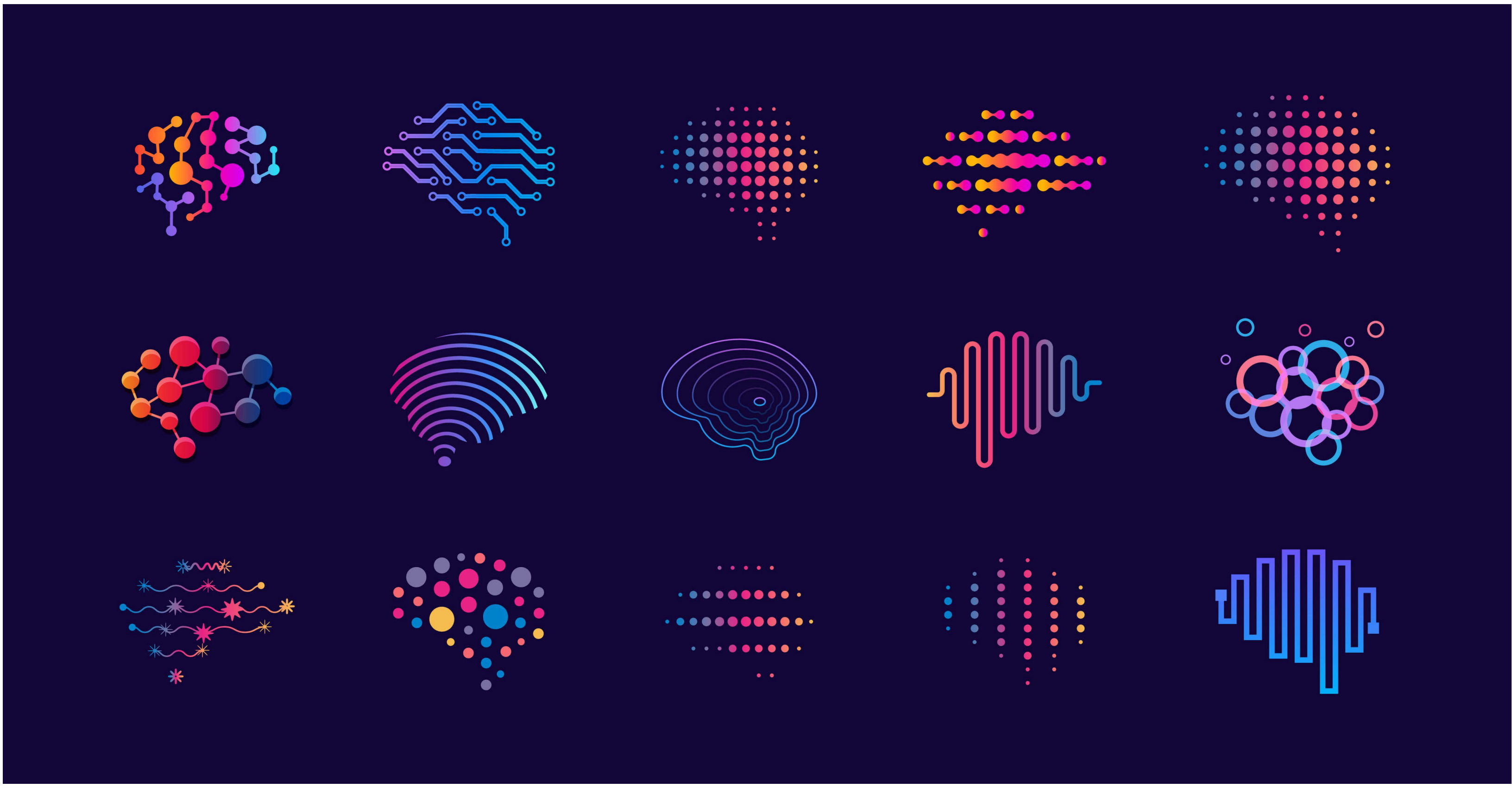
ANL-23/15

2022 AI Testbed Expeditions Report

Laboratory Directed Research and Development (LDRD)

Advanced Computing Expedition Leads:
Valerie Taylor, Ian Foster, Salman Habib, and Michael E. Papka

Computing, Environment and Life Sciences Directorate



About Argonne National Laboratory
Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free at OSTI.GOV (http://www.osti.gov/), a service of the U.S. Dept. of Energy's Office of Scientific and Technical Information

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703)
605-6000 Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

ANL-23/15

2022 AI Testbed Expeditions Report

Laboratory Directed Research and Development (LDRD)

prepared by
Venkat Vishwanth and Murali Emani

Computing, Environment and Life Sciences Directorate, Argonne National Laboratory

December 2022

Authors

Anakha V. Babu, Henry Chan, Mathew J. Cherukara, Jose M. Monsalve Diaz, Johannes Doerfert, Jeremy Feinstein, Ian Foster, Ross J. Harder, Kevin Hickey, Jan Hückelheim, Saugat Kandel, Rajkumar Kettimuthu, Tadbhagya Kumar, Zhengchun Liu, Margaret MacDonell, Antonino Miceli, Marieme Ngom, Pinaki Pal, Noah Paulson, Kurt Picel, Krishnan Raghavan, Arvind Ramanathan, Esteban Rangel, Siddhisanket Raskar, Varuni Sastry, Ganesh Sivaraman, Baixi Sun (Indiana University), Marco Trovato, Lauren Valentino, Zhen Xie, Eugene Yan, Yudong Yao, Kazutomo Yoshii, Xiaodong Yu, Tao Zhou

Table of Contents

AI accelerator for scalable DL-based 3D X-ray nanoscale imaging

Henry Chan (Assistant Scientist, CNM), Yudong Yao (Assistant Physicist, APS),
Varuni Sastry (Predoctoral appointee, ALCF) Ross J. Harder (Physicist, APS),
Mathew J. Cherukara (Computational Scientist, APS)

November 2022

1 Introduction to the Science problem

By exploiting the coherent properties of a light source, coherent diffraction imaging (CDI) is able to obtain the sample image at a nanoscale resolution using the measured diffraction pattern. Bragg Coherent Diffraction Imaging (BCDI) has become valuable for recovering the displacement and strain field of crystals, providing a valuable tool in material science and solid-state physics. X-ray ptychography is another emerging CDI technique that can produce a high-resolution image of the extended sample and has has become popular in many research areas (e.g., materials science, biology, electronics, and optics characterization). CDI including BCDI and ptychography has become an established technique in Synchrotron Facilities including the Advanced Photon Source (APS) and will greatly benefit from the 100× coherent flux increase of the upcoming APS Upgrade (APSU). The current image formation process in CDI employs iterative phase retrieval algorithms, which is a time-consuming and computationally expensive process. Especially after APSU, the traditional iterative methods will not be able to match the experimental data acquisition speed. We employ deep learning (DL) approach to replace the iterative approaches, therefore allowing hundreds of times faster recovery of the object.

We developed AutoPhaseNN, a DL-based approach which learns to solve the inverse problem without labeled data. Taking 3D BCDI as a representative technique, AutoPhaseNN has been demonstrated to be one hundred times faster than traditional iterative phase retrieval methods while providing comparable image quality. The current network is trained with $64 \times 64 \times 64$ data size, to achieve higher resolution imaging, we will need to scale the network to input and train/infer 3D arrays of size $256 \times 256 \times 256$ (today) and of size $2560 \times 2560 \times 2560$ (APSU). However, the scalability of the network is restricted due to the memory-intensive training process. To perform the training for a $256 \times 256 \times 256$ data size, the required memory exceeds the capacity of the current machine. In this project, we explore using Sambanova system to train the network for the direct data inversion for CDI.

2 Description of the AI model and implementation

AutoPhaseNN is an unsupervised physics-aware deep learning model, which is built by combining a conventional 3D convolutional neural network (CNN) with the physical model of X-ray scattering. The network is trained with only simulated and/or measured diffraction patterns without needing real space images. Once trained, the physical model can be discarded and only the 3D CNN model is used to provide the inversion from 3D diffraction intensities to real space images.

The architecture of AutoPhaseNN is depicted in Fig. 1. The model is based on a 3D CNN framework with an encoder-decoder architecture. The encoder takes the magnitude of the 3D diffraction pattern with the size of $64 \times 64 \times 64$ as its input. Then, the encoded data is passed through two separated decoders to generate the $64 \times 64 \times 64$ amplitude and phase images in real space. The input is connected to the output using convolution blocks, max pooling, upsampling and zero padding layers. The convolution block is composed of two 3×3×3 convolution layers, the leaky rectified linear unit (LRLU) activation function and the batch normalization (BN). The physical knowledge is enforced via the Sigmoid and Tanh activation functions in the final layers. In addition to the 3D CNN portion, we include the X-ray scattering model into the network architecture, as shown in Fig. 1B. The outputs of the 3D CNN, amplitude and phase images, are combined to form the complex number. An image shape support function is obtained by thresholding the current predicted amplitude using a contour at the 10% intensity level. The estimated diffraction pattern is obtained from the 3D complex Fourier transform of the current estimation of the real space image. The network weight and bias factors are optimized with the objective function that minimizes the loss between the input diffraction pattern and the estimated diffraction pattern. By incorporating the physics, the ground truth of the real space image is not needed during training. Once a trained model has been obtained, only the 3D CNN model is kept to recover the amplitude and phase information of the measured sample.

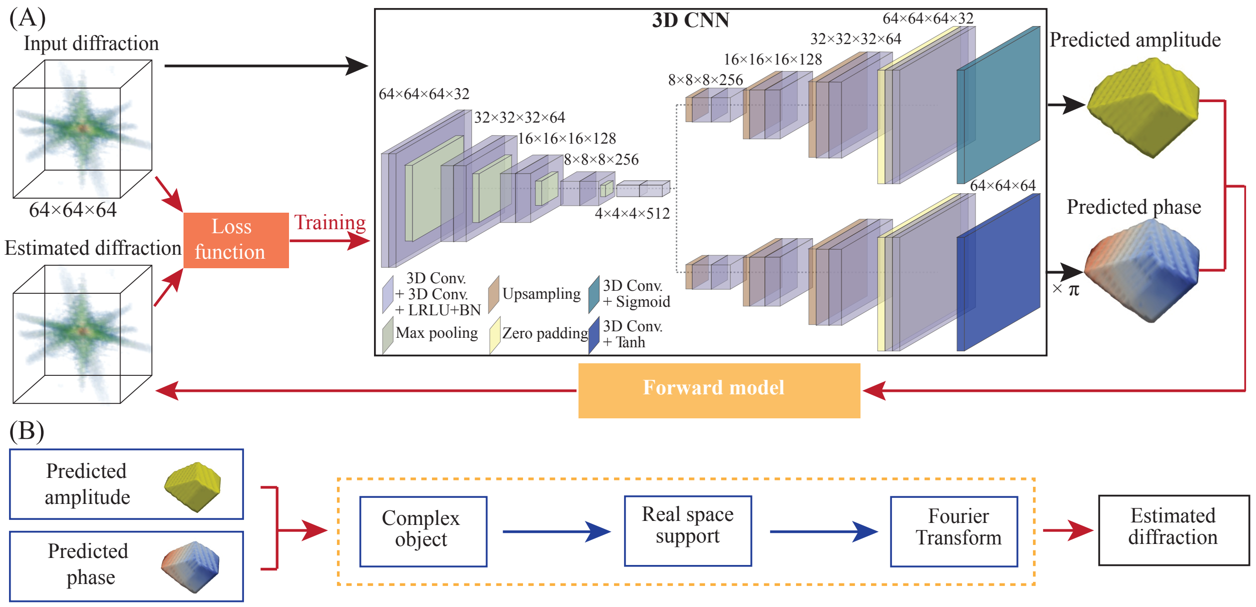


Figure 1: Schematic of the neural network structure of AutoPhaseNN during training. (**A**) The model consists of a 3D CNN and the X-ray scattering forward model. (**B**) The X-ray scattering forward model.

3 What was needed to get the model running on the AI Accelerator

The network model was originally implemented based on the TensorFlow 2.0 platform and we have converted it to PyTorch. As shown in Fig. 1 (A), the functions including convolution, batch normalization, maxpooling, upsampling, and zeropadding for 3D convolutional neural network, as well as the LRLU, Sigmoid, and Tanh activation fucntions are needed. Additionally, the 3D complex Fourier transform is needed in the forward model to realize the unsupervised learning, as shown in Fig. 1(B). Based on the discussion with the SambaNova team, some functions for the 3D nueral network, especially the 3D complex Fourier transform are not available on SambaNova system at this time, so we started with implementing a 2D supervised network (PtychoNN), which is a deep convolutional neural network developed for the ptychography data inversion.

Figure 2 shows the structure of PtychoNN, which has similar network architecture with the 3D CNN in AutoPhaseNN. The network takes the X-ray diffraction data (64 $\times$ 64 sized array) as input and outputs both real space amplitude and phase images. The neural network consists of 3 parts, the encoder arm that learns a representation in feature space of the input X-ray diffraction data and two decoder arms that learn to map to real-space amplitude and phase, respectively. The difference between PtychoNN and AutoPhaseNN is that it is a supervised neural network, so the labeled data, pairs of diffraction pattern and real-space images are needed for network training. But the forward process, which is the 2D complex Fourier transform for ptychography, is not required.

Currently, most of the functions needed in the PtychoNN are supported by SambaNova, except for

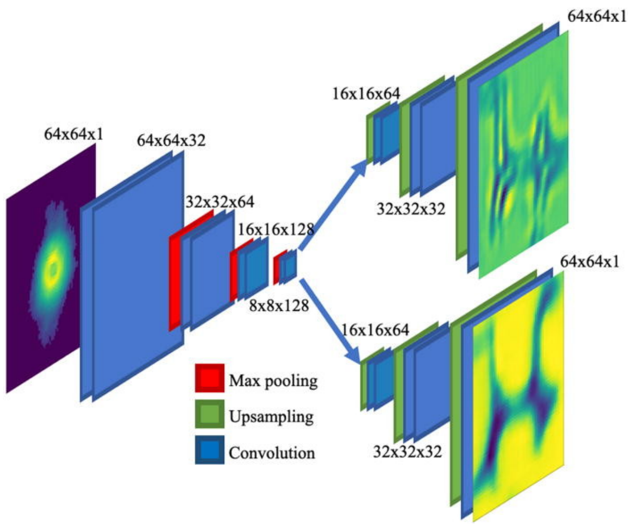


Figure 2: Architecture of PtychoNN, a 2D supervised neural network that can predict real-space amplitude and phase from input diffraction data alone.

upsampling, which can be replaced by ConvTranspose2d with the stride being set to 2. We have successfully implemented the PtychoNN on SamboNova.

4 Performance Evaluation

To evaluate the performance of PtychoNN on SambaNova machine, we first trained the network on TheteGPU for comparison. The training data consists of 20608 pairs of diffraction pattern and the corresponding real-space amplitude and phase images, within which 640 pairs are reserved for validation. When training the model, the weights and biases of the model is updated to minimize per-pixel mean absolute error (MAE). Weight updates are made using AdamW optimization method with the learning rate of 0.001. At the end of each epoch, the performance of the network was evaluated using the validation dataset. Then the same training processes were repeated on SambaNova machine.

Batch size	**Loss**	**ThetaGPU**	**SambaNova**
32	Train	0.0709	0.0755
	Validation	0.0900	0.1000
64	Train	0.0681	0.0719
	Validation	0.1000	0.1000
128	Train	0.0643	0.0673
	Validation	0.1100	0.1100

Table 1: Comparison of training and validation losses for PtychoNN on ThetaGPU and SambaNova

Table 1 shows the training and validation losses (MAE) for PtychoNN trained on single A100 GPU on ThetaGPU and trained on SambaNova machine after 100 epochs, showing that the training and validation losses obtained on SambaNova are comparable to the losses got on ThetaGPU.

Then we compare the training speed of PtychoNN. Table 2 shows that training on SambaNova is about 35%, 20%, 14% faster with batch size of 32, 64, and 128, respectively, compared to training on ThetaGPU. The observation that the speed improvement is larger with smaller batch size implies that the data I/O on

Batch size	**ThetaGPU** (s/epoch)	**SambaNova** (s/epoch)
32	(A) 10.052	(A) 6.561
	(T) 9.694	(T) 6.282
	(V) 0.357	(V) 0.278
64	(A) 7.600	(A) 6.137
	(T) 7.239	(T) 5.870
	(V) 0.361	(V) 0.268
128	(A) 6.614	(A) 5.668
	(T) 6.332	(T) 5.411
	(V) 0.282	(V) 0.256

Table 2: Speed comparison for PtychoNN training on ThetaGPU and on SambaNova, where A stands for the total time, T is the training time per epoch, and V is the validation time per epoch.

SambaNova is more efficient than on ThetaGPU. Further profiling of the model will be needed to better understand this.

Batch size	**w/o SambaLoader** (s/epoch)	**w/ SambaLoader** (s/epoch)	**MACv2 compiler** (s/epoch)
32	6.204	5.950	7.771
64	5.622	5.301	7.189
128	5.094	4.945	6.935

Table 3: Training speed of PtychoNN on SambaNova with the SambaLoader function and mac-v2 compiler

According to the discussion with the SambaNova team, the performance of the network may be improved by using the SambaLoader in the data loading stage. The SambaLoader assists with parallelizing load operations alongside graph operations, while also automatically converting torch tensors to SambaTensors before the training process started. The training time for PtychoNN network with and without using SambaLoader is shown in Table 3. The training time needed for each epoch can be reduced by $3 \sim 5\%$ with the SambaLoader function. Another thing we tried is using the 2nd version of the SambaNova compiler (MACv2) to perform the network compiling. One thing we noticed is that the MACv2 didn't work for MAE loss (L1Loss), so we replace it with the mean absolute error (MSE). The time information using MACv2 compiler is also in Table 3, showing that it was a little bit slower than the MACv1 compiler. The SambaNova team mentioned that MACv2 was still being worked on, so maybe there were some gaps in the performance, and they have started a deeper investigation into the behaviour of MACv2.

5 AutophaseNN implementation on SN

The 3D implementation is in development by the SN engineering team. They have implemented the model for an input size of 32^3. The figure 3 shows the training loss with the SN implementation on RDU (refer to the plot in blue). The implementation called "OurGold" differs from the original PyTorch model in using ConvTranspose3D instead of the Upsampling and torch.relu subtraction substituting torch.where. It showcases a similar loss trends when compared with the CPU/GPU implementation (shown in orange/blue). The difference in the loss can be attributed to the lower precision used on the SN. An implementation for 64^3 which is the minimum required for practical application is still ongoing.

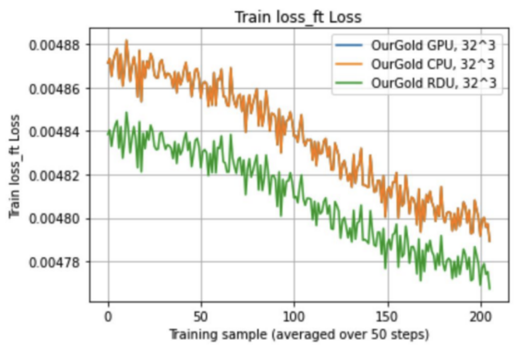


Figure 3: training loss of the SN autophasenn implementation of the 32x32x32 input image.

6 Conclusion and next steps

In conclusion, we have successfully implemented PtychoNN, the 2D deep convolutional neural network, on SambaNova machine. The loss calculated on training and validation datasets when trained on SambaNova is comparable to the loss acquired on ThetaGPU. Given different batch sizes, the training speed on SambaNova is about 39%, 28%, 22% faster than ThetaGPU. To better understand the performance variation for different batch sizes, further profiling of the model is needed.

For the next step, we plan to work on the 3D unsupervised neural network, AutoPhaseNN. The PyTorch functions, e.g. 3D convolution, upsampling, maxpooling and zeropadding, as well as the 3D complex Fourier transform are essential to make AutoPhaseNN work on SambaNova. Once these operations are fully supported by SN and AutoPhaseNN can be trained on SambaNova, we will work on a larger network with 3D array size of $256 \times 256 \times 256$ to meet the current requirement of 3D BCDI data analysis, then further scale the network to train/infer 3D arrays of size $2560 \times 2560 \times 2560$ for BCDI after the APS upgrade.

7 Acknowledgements

We thank Varuni Sastry of Argonne Leadership Computing Facility, for getting the model work on SambaNova system. We also thank Saugat Kandel of Advanced Photon Source, for implementing the ConvTranspose2d in PtychoNN.

This work was performed, in part, at the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science User Facility, operated for the DOE Office of Science by Argonne National Laboratory under Contract No. DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

Predicting degradation of microplastics with a deep learning approach

Jeremy Feinstein, Kevin Hickey, Ganesh Sivaraman, Eugene Yan, Margaret MacDonell, Kurt Picel, Arvind Ramanathan and Ian Foster

October 14, 2022

1 Introduction to the Science problem

The accumulation of plastic waste in the environment is a growing problem for all sectors of industry. Currently, more than 400 million tons of plastics are produced globally each year, with 50 million tons coming from the U.S. alone. Efforts to mitigate the discharge and accumulation of these materials through recycling programs have been insufficient as recycling rates remain under 10% both nationally and globally. Overall, 70% of the plastic that has ever been produced (more than 9 billion tons) remains in the environment where it threatens biodiversity and ecosystem services through its long-term persistence, ability to absorb pollutants, ease of biouptake and potential to fragment into highly mobile nano-scale debris. In an effort to diminish these harmful environmental effects, an effort is underway to design and develop polymer materials with more favorable degradation properties. To accomplish this it is necessary to develop a framework capable of quantifying and predicting a polymer's end-of-life (EOL) environmental degradation properties from its molecular and bulk level materials characteristics. Given the complexity of the problem however, with multiple environmental EOL compartments (soil, sediment, marine and freshwater) and degradation pathways (mechanical, thermal, photo, and biological), attempts to quantify long-term degradation properties through laboratory measurements alone have been insufficient.

Artificial intelligence and machine learning (AI/ML) techniques provide an opportunity to bridge these gaps though coupling complex neural networks with expansive feature datasets to discover relationships between polymer material properties and EOL effects. These datasets include both bulk polymer properties like molecular weight, enthaply of melting and degree of crystallinity, as well as molecular properties like atom type and valence, bond hybridization and resonance effects. The properties selected for prediction in this study were glass transition temperature, T_g, melting temperature, T_m, and density, ρ. These properties are critical to quantifying how a given polymer reacts to the thermal and mechanical forces of environmental degradation and are thus necessary precursors to more broad EOL effects.

2 Description of the AI model and implementation

Datasets. We focus on homopolymers as they are conveniently represented by their constituting monomer unit. Data is organized into 3 property labels (T_g, T_m, ρ) aggregated from multiple sources [1–84]. If not provided, SMILES are generated from IUPAC names, common names, or structural identifiers via the online CACTUS [85] and OPSIN tools [86]. All SMILES are then canonicalized using RDKit [87]. If multiple labels are given for one structure they are treated as replicate experiments and the property is estimated via geometric mean. For each dataset, property counts are: 7,558 glass transition temperatures; 3,730 melting temperatures; and 1,889 densities. The distributions for each dataset are shown in figure 1 and are described as approximately normal.

Graph Convolutional Neural Networks [88]. Two model architectures are explored for this problem: (1) Graph Convolutional Neural Network (GCN) [88], and (2) Message Passing Neural Network (MPNN) [89]. GCNs learn on atoms attributes via features aggregated across neighboring atoms. The layer-wise propagation rule for layer $l + 1$ at node v_i is given by:

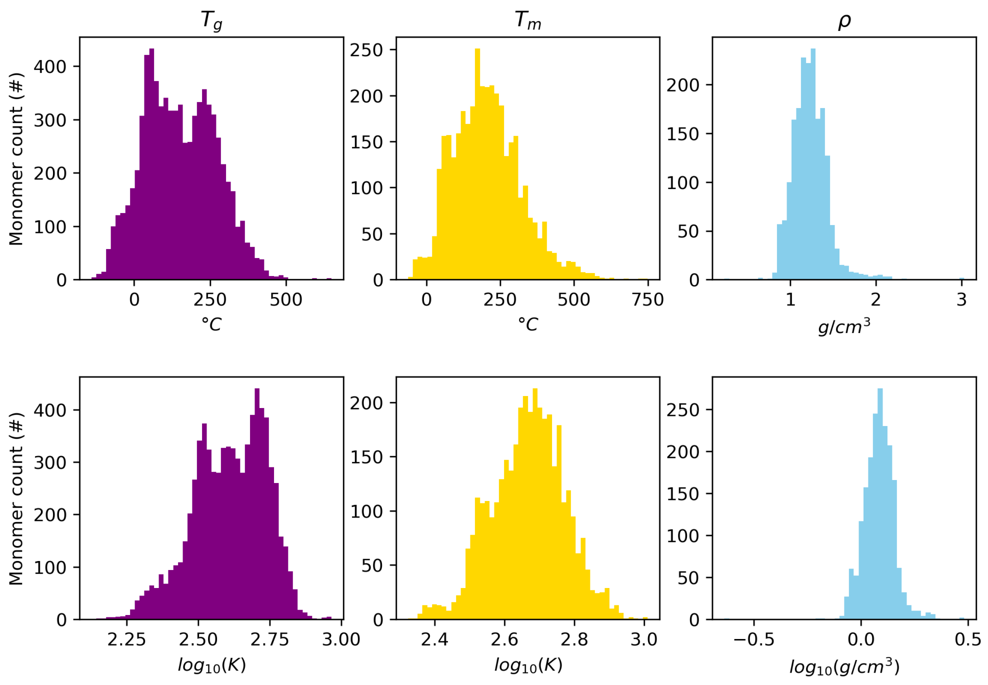


Figure 1: Distribution of property labels in glass transition temperature, T_g, melting temperature, T_m, and density, ρ, datasets. Real (top row) and log-transformed (bottom row) units are used to demonstrate transformation affect on distribution shape.

$$h_{v_i}^{l+1} = \sum_{v_j \in \mathcal{N}(v_i)} \frac{1}{c_{ij}} h_{v_j}^{l} W^{l}, \tag{1}$$

where W^l is the l-th layer weight matrix, and $c_i j$ is a normalization constant calculated by symmetrically normalizing the adjacency matrix after removing self-loops.

Message Passing Neural Networks [89]. GCNs alone cannot produce feature maps for explainable AI (XAI) analysis against current domain models which rely on quantum descriptors calculated at the bond-level. For this we employ MPNNs which learn from bonds informed by neighboring atom features via multilayer perceptron. The formulation for MPNN changes to:

$$h_{v_i}^{l+1} = w h_{v_i}^{l} + \sum_{j \in \mathcal{N}(v_i)} w_{\Theta}(h_{v_i}^{l}, h_{v_j}^{l}, e_{ij}), \tag{2}$$

where w is the edge-specific weight, e_{ij} are edge features, and w_Θ is a neural network (e.g., multilayer perceptron).

Heat maps. The result of layer-wise propagation algorithms can be applied in multiple ways to produce heat maps. The *class activation map* (CAM) method uses the product of the activations of the last convolutional layer H^L with the global add pool (GAP) layer $\sum_{i \in features} H_i^L$ with minimal extra computation or code required [90]. In *gradient CAMs* (GradCAM), heat maps are approximated using the gradients of the neural network output, $\hat{y}$, with respect to the GAP layer [90]. The embeddings for several important polymers are selected to compare against QC calculations.

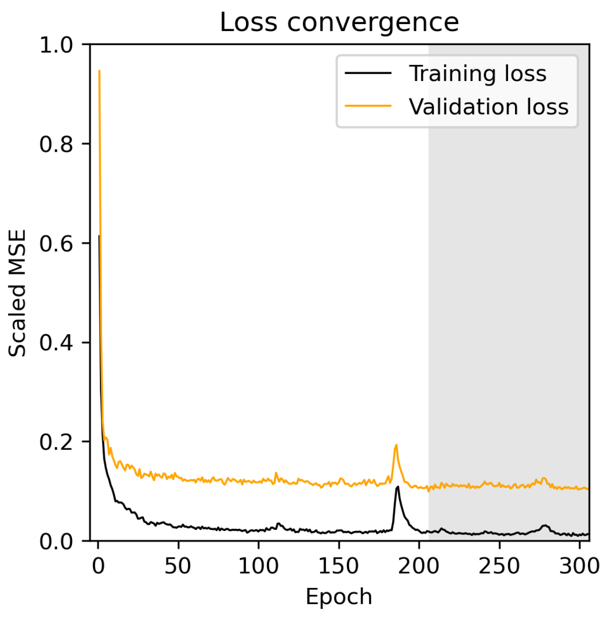


Figure 2: Loss convergence of a GCN on NVIDIA A100 for a single fold of the T_g dataset. Training (black) and validation (orange) loss are shown for comparison. Training is complete after 306 epochs when validation loss hasn't decreased below the global minimum for 100 epochs. The last 100 epochs (grey shading) are thrown out and weights are restored.

3 What was needed to get the model running on the AI Accelerator

The GCN and MPNN model implemented in the PyTorch Geometric extension (PyG) for PyTorch [91, 92] on NVIDIA A100 served as the initial starting point for porting models to SambaNova. Also, a separate GCN implementation with no PyG dependency was implemented on NVIDIA A100 [93]. SambaNova API codes from the example files pointed to in the user tutorials were adjusted to work with our codes.

On the SambaNova testbed, errors are specific but lack documentation for guidance. The PyG models failed to work at the installation step, since we could not get a working PyG package running on SambaNova. The vanilla Torch model failed at the compilation step, as the required matrix multiplication operation was not compatible with SambaNova's RDU. We also tried batch dense multiplication instead of the standard matrix multiplication.

Attempts at further implementing RDU codes for SambaNova were stifled by lack of documentation in the API. The SambaNova technical team was receptive to our requests for help, but indicated we had to choose one model for which we could recieve assistance by end of September deadline. The team is currently working on implementing our PyG GCN and it is not complete yet.

4 Performance Evaluation

No performance metrics for training time and and accuracy are available for the SambaNova accelerator. Benchmark models developed for 1x NVIDIA A100 with 2x 64-core AMD EPYC 7742 were trained using the Swing HPC at LCRC. Loss convergence and prediction plots of T_g for the best performing GCN are shown in Figures 2 and 3. Table 1 shows performance metrics for a GCN on T_g for each cross-validated testing fold. On NVIDIA A100, the model trained for an average of 0.5386 s/epoch (23.41 ms/step).

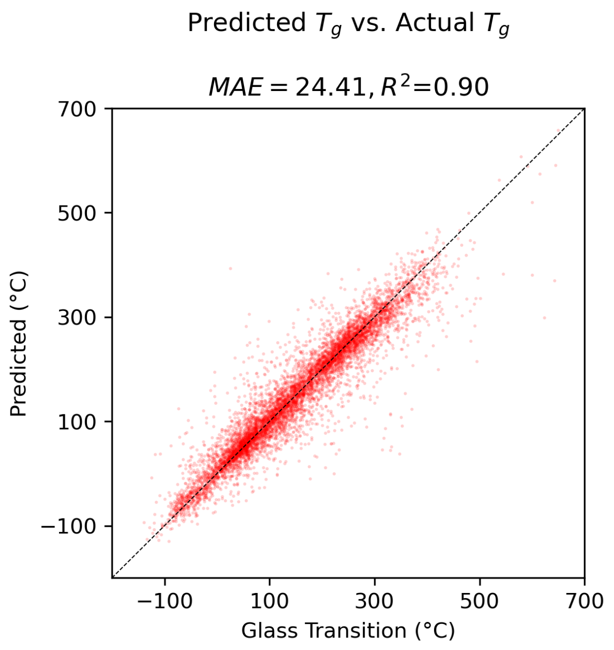


Figure 3: GCN performance for T_g on NVIDIA A100 aggregated across all validation folds. The model predicts with a strong correlation (average fold $R^2 = 0.90$) when compared to true T_g.

Fold	MAE	RMSE	R2
1	23.68	37.41	0.90
2	23.83	36.40	0.90
3	24.04	37.29	0.90
4	25.54	40.58	0.88
5	24.94	38.74	0.90
Avg.	24.41	38.08	0.90

Table 1: Performance metrics of GCN on NVIDIA A100 for each T_g validation fold.

5 Conclusion and next steps

We are currently waiting for the SambaNova assistance team to provide guidance on how to convert our Torch PyG model with compatibility on NVIDIA A100 to the SambaNova RDU. The major lesson learned is that dense multiplication is very challenging without structural inefficiencies like those on GPU/CPU. In that case, depending on dataset size and the number of models that need to be tested, the consequences of time-to-train on GPU may not outweigh the consequences of time-to-implement on novel processor architectures. These questions will continue to be asked.

The work developed during this LDRD will be prepared for a manuscript publication with additional results for homopolymer melting temperature, density, and XAI using the heat map methodologies described previously.

Next steps. The next steps in our model development consist first and foremost of implementing and finalizing the GCN and MPNN models on the Samba Nova testbed with the assistance of the Samba Nova team. Once completed, those models will be compared to complementary models for our predicted properties developed with quantum chemistry (QC) methods to determine how these two techniques can be be best utilized moving forward. Finally, with a proven model framework we'll begin investigating more complex polymer EOL properties that more fully describe and quantify the broad effects of environmental polymer degradation.

6 Acknowledgements

We gratefully acknowledge the user of the SambaNova AI testbed operated by the Argonne Leadership Computing Facility and the high-performance computing cluster Swing operated by the Laboratory Computing Resource Center at Argonne National Laboratory, both of which are DOE Office of Science User Facilities supported under Contract DE-AC02-06CH11357.

We thank the SambaNova technical team for ongoing assistance working to make our models compatible on the testbed.

We also gratefully acknowledge the work of three SULI students: Scott Coia, who aggregated and processed datasets; Carlos Matherson, who programmed initial GCNs in TensorFlow that served as a guide for our Torch implementation; and Jason Xu, who contributed to the Torch implementation and repository structure.

This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.

References

[1] Shingo Otsuka et al. "PoLyInfo: Polymer Database for Polymeric Materials Design". In: *2011 International Conference on Emerging Intelligent Data and Web Technologies*. 2011, pp. 22–29. DOI: 10.1109/EIDWT.2011.13.

[2] Lanhe Zhang et al. "Major impact of cyclic chain topology on the tg-confinement effect of supported thin films of polystyrene". en. In: *Macromolecules* 49.1 (Jan. 2016), pp. 257–268.

[3] Weiyu Wang et al. "High temperature thermoplastic elastomers synthesized by living anionic polymerization in hydrocarbon solvent at room temperature". en. In: *Macromolecules* 49.7 (Apr. 2016), pp. 2646–2655.

[4] Xiaoxiao Wang, Christian Pellerin, and C Geraldine Bazuin. "Enhancing the electrospinnability of low molecular weight polymers using small effective cross-linkers". en. In: *Macromolecules* 49.3 (Feb. 2016), pp. 891–899.

[5] Ashish Batra et al. "Counterion effect on the rheology and morphology of tailored poly(dimethylsiloxane) ionomers". en. In: *Macromolecules* 39.4 (Feb. 2006), pp. 1630–1638.

[6] Katrien V Bernaerts et al. "PH-responsive diblock copolymers prepared by the dual initiator strategy". en. In: *Macromolecules* 39.11 (May 2006), pp. 3760–3769.

[7] Toshiyuki Fukushima, Kiyohiro Takachi, and Kenji Tsuchihara. “Optically active poly(phenylacetylene) film: Simultaneous change of color and helical structure”. en. In: *Macromolecules* 39.9 (May 2006), pp. 3103–3105.

[8] Douglas R Robello et al. “Polymers for photothermography: Controlled thermal release of color photographic developer from substituted polystyrene derivatives”. en. In: *Macromolecules* 39.17 (Aug. 2006), pp. 5686–5695.

[9] Tadahisa Iwata et al. “Microbeam X-ray diffraction and enzymatic degradation of poly[(R)-3-hydroxybutyrate] fibers with two kinds of molecular conformations”. en. In: *Macromolecules* 39.17 (Aug. 2006), pp. 5789–5795.

[10] 2008.

[11] Takashi 2baki. *Thymus activation healing.* Independently Published, Sept. 2022.

[12] J. Bicerano. *Prediction of Polymer Properties.* Plastics Engineering. CRC Press, 2002. ISBN: 9780203910115. URL: https://books.google.com/books?id=5L1USsiq6KcC.

[13] Suneel Bandi and David A Schiraldi. “Glass transition behavior of clay aerogel/poly(vinyl alcohol) composites”. en. In: *Macromolecules* 39.19 (Sept. 2006), pp. 6537–6545.

[14] L Sauguet et al. “Synthesis and characterization of poly(vinylidene fluoride)-g-poly(styrene) graft polymers obtained by atom transfer radical polymerization of styrene”. en. In: *Macromolecules* 39.26 (Dec. 2006), pp. 9087–9101.

[15] E Bhoje Gowd, Naoya Shibayama, and Kohji Tashiro. “Structural changes in thermally induced phase transitions of uniaxially oriented δe form of syndiotactic polystyrene investigated by temperature-dependent measurements of X-ray fiber diagrams and polarized infrared spectra”. en. In: *Macromolecules* 39.24 (Nov. 2006), pp. 8412–8418.

[16] V Khanna et al. “Chain architecture effects on the diffusion of cylinder-forming block copolymers”. en. In: *Macromolecules* 40.7 (Apr. 2007), pp. 2443–2452.

[17] Mahesh K Mahanthappa et al. “Control of mechanical behavior in polyolefin composites: Integration of glassy, rubbery, and semicrystalline components”. en. In: *Macromolecules* 40.5 (Mar. 2007), pp. 1585–1593.

[18] Tianqi Liu et al. “Synthesis of polymandelide: A degradable polylactide derivative with polystyrene-like properties”. en. In: *Macromolecules* 40.17 (Aug. 2007), pp. 6040–6047.

[19] Samuel Merlet et al. “Preparation and characterization of nanocellular poly(phenylquinoxaline) foams. A new approach to nanoporous high-performance polymers”. en. In: *Macromolecules* 40.6 (Mar. 2007), pp. 2070–2078.

[20] T Y Cho, W Stille, and G Strobl. “Zero growth temperature of crystallizing polyethylene”. en. In: *Macromolecules* 40.7 (Apr. 2007), pp. 2596–2599.

[21] Tatjana Haramina and Reiner Kirchheim. “Mechanical spectroscopy of PVCN with increasing cross-linking degree”. en. In: *Macromolecules* 40.12 (June 2007), pp. 4211–4216.

[22] Antti Nykänen et al. “Phase behavior and temperature-responsive molecular filters based on self-assembly of polystyrene-block-poly(N-isopropylacrylamide)-block-polystyrene”. en. In: *Macromolecules* 40.16 (Aug. 2007), pp. 5827–5834.

[23] Gang Liu et al. “Graph Rationalization with Environment-based Augmentations”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* Association for Computing Machinery, 2022, pp. 1069–1078.

[24] Bhaskar Sharma et al. “Enzymatic synthesis and solid-state properties of aliphatic polyesteramides with polydimethylsiloxane blocks”. en. In: *Macromolecules* 40.22 (Oct. 2007), pp. 7919–7927.

[25] Hironori Kaji and Fumitaka Horii. “Investigation of dynamics of poly(dimethylsilane) in the mesophase by solid-state 29Si NMR: Evidence for rotator phase”. en. In: *Macromolecules* 40.15 (July 2007), pp. 5420–5423.

[26] Hong Chen, Joshua D Snyder, and Yossef A Elabd. “Electrospinning and Solution Properties of Nafion and Poly(acrylic acid)”. en. In: *Macromolecules* 41.1 (Jan. 2008), pp. 128–135.

[27] D J M van Beek et al. “Unidirectional dimerization and stacking of ureidopyrimidinone end groups in polycaprolactone supramolecular polymers”. en. In: *Macromolecules* 40.23 (Nov. 2007), pp. 8464–8475.

[28] Nam-Goo Kang, Mohammad Changez, and Jae-Suk Lee. “Living anionic polymerization of the amphiphilic monomer 2-(4-vinylphenyl)pyridine”. en. In: *Macromolecules* 40.24 (Nov. 2007), pp. 8553–8559.

[29] Yun Suk Jo et al. “RAFT homo- and copolymerization of N-acryloyl-morpholine, piperidine, and azocane and their self-assembled structures”. en. In: *Macromolecules* 41.4 (Feb. 2008), pp. 1140–1150.

[30] E Bhoje Gowd, Naoya Shibayama, and Kohji Tashiro. “Structural correlation between crystal lattice and lamellar morphology in the phase transitions of uniaxially oriented syndiotactic polystyrene (δ and δe forms) as revealed by simultaneous measurements of wide-angle and small-angle X-ray scatterings”. en. In: *Macromolecules* 41.7 (Apr. 2008), pp. 2541–2547.

[31] Frédéric Boschet et al. “Radical copolymerization of α,β-difluoroacrylic acid with vinylidene fluoride”. en. In: *Macromolecules* 43.11 (June 2010), pp. 4879–4888.

[32] Virginie M Boucher et al. “Enthalpy recovery of PMMA/silica nanocomposites”. en. In: *Macromolecules* 43.18 (Sept. 2010), pp. 7594–7603.

[33] Wenchun Fan, Lei Wang, and Sixun Zheng. “Double reaction-induced microphase separation in epoxy resin containing polystyrene-block-poly(ϵ-caprolactone)-block-poly(n-butyl acrylate) ABC triblock copolymer”. en. In: *Macromolecules* 43.24 (Dec. 2010), pp. 10600–10611.

[34] *PolymerDatabase*. `https://www.polymerdatabase.com`. Online; accessed 2022.10.10.

[35] Dominic Valiquette and Christian Pellerin. “Miscible and Core-Sheath PS/PVME fibers by electrospinning”. en. In: *Macromolecules* 44.8 (Apr. 2011), pp. 2838–2843.

[36] Siti Nurkhamidah and Eamor M Woo. “Phase separation and lamellae assembly below UCST in poly(l-lactic acid)/poly(1,4-butylene adipate) blend induced by crystallization”. en. In: *Macromolecules* 45.7 (Apr. 2012), pp. 3094–3103.

[37] Hongjun Yang et al. “Hybrid copolymerization of ϵ-caprolactone and methyl methacrylate”. en. In: *Macromolecules* 45.8 (Apr. 2012), pp. 3312–3317.

[38] Jan Seuring and Seema Agarwal. “First example of a universal and cost-effective approach: Polymers with tunable upper critical solution temperature in water and electrolyte solution”. en. In: *Macromolecules* 45.9 (May 2012), pp. 3910–3918.

[39] Yuanrong Cheng et al. “Synthesis and properties of highly cross-linked thermosetting resins of benzocyclobutene-functionalized benzoxazine”. en. In: *Macromolecules* 45.10 (May 2012), pp. 4085–4091.

[40] Yoseop Kim et al. “Comparison of shell-cross-linked micelles with soft and glassy cores as a drug delivery vehicle for albendazole: Is there a difference in performance?” en. In: *Macromolecules* 45.13 (July 2012), pp. 5451–5462.

[41] Miao Hong et al. “Synthesis of novel cyclic olefin copolymer (COC) with high performance via effective copolymerization of ethylene with bulky cyclic olefin”. en. In: *Macromolecules* 45.13 (July 2012), pp. 5397–5402.

[42] Kaewkan Wasanasuk and Kohji Tashiro. “Theoretical and experimental evaluation of crystallite moduli of various crystalline forms of poly(l-lactic acid)”. en. In: *Macromolecules* 45.17 (Sept. 2012), pp. 7019–7026.

[43] Deepika Rangari and Nadarajah Vasanthan. “Study of strain-induced crystallization and enzymatic degradation of drawn poly(l-lactic acid) (PLLA) films”. en. In: *Macromolecules* 45.18 (Sept. 2012), pp. 7397–7403.

[44] Csaba Fodor et al. “Thermal behavior, stability, and decomposition mechanism of poly(N-vinylimidazole)”. en. In: *Macromolecules* 45.22 (Nov. 2012), pp. 8953–8960.

[45] Donald J Darensbourg et al. "Dramatic behavioral differences of the copolymerization reactions of 1,4-cyclohexadiene and 1,3-cyclohexadiene oxides with carbon dioxide". en. In: *Macromolecules* 48.6 (Mar. 2015), pp. 1679–1687.

[46] Toshinori Fujie et al. "Selective molecular permeability induced by glass transition dynamics of semicrystalline polymer ultrathin films". en. In: *Macromolecules* 46.2 (Jan. 2013), pp. 395–402.

[47] Christopher M Evans et al. "Simultaneous determination of critical Micelle temperature and Micelle core glass transition temperature of block copolymer–solvent systems via pyrene-label fluorescence". en. In: *Macromolecules* 46.10 (May 2013), pp. 4131–4140.

[48] Koffi L Dagnon et al. "Controlling the rate of water-induced switching in mechanically dynamic cellulose nanocrystal composites". en. In: *Macromolecules* 46.20 (Oct. 2013), pp. 8203–8212.

[49] Zhiyong Jiang et al. "Tensile Deformation of Oriented Poly(ϵ-caprolactone) and Its Miscible Blends with Poly(vinyl methyl ether)". en. In: *Macromolecules* 46.17 (Sept. 2013), pp. 6981–6990.

[50] Katherine P Barteau et al. "Allyl glycidyl ether-based polymer electrolytes for room temperature lithium batteries". en. In: *Macromolecules* 46.22 (Nov. 2013), pp. 8988–8994.

[51] Won Jae Yoon et al. "Synthesis and characteristics of a biobased high-tg terpolyester of isosorbide, ethylene glycol, and 1,4-cyclohexane dimethanol: Effect of ethylene glycol as a chain linker on polymerization". en. In: *Macromolecules* 46.18 (Sept. 2013), pp. 7219–7231.

[52] Hafiz Ashraful Haque et al. "Photoinduced in-plane motions of azobenzene mesogens affected by the flexibility of underlying amorphous chains". en. In: *Macromolecules* 46.20 (Oct. 2013), pp. 8275–8283.

[53] Daisuke Kawakami et al. "New insights into lamellar structure development and SAXS/WAXD sequence appearance during uniaxial stretching of amorphous poly(ethylene terephthalate) above glass transition temperature". en. In: *Macromolecules* 41.8 (Apr. 2008), pp. 2859–2867.

[54] Paul Painter and He Huang. "Concerning the determination of conformational states in polymers by infrared spectroscopy: A study of poly(methyl methacrylate)". en. In: *Macromolecules* 41.7 (Apr. 2008), pp. 2494–2501.

[55] Masatoshi Shioya et al. "Small-angle X-ray scattering study on the tensile fracture process of poly(ethylene terephthalate) fiber". en. In: *Macromolecules* 41.13 (July 2008), pp. 4758–4765.

[56] Masami Sano et al. "New aspects for the hierarchical cooperative motions in photoalignment process of liquid crystalline block copolymer films". en. In: *Macromolecules* 48.7 (Apr. 2015), pp. 2217–2223.

[57] Shingo Kobayashi et al. "Synthesis and properties of new thermoplastic elastomers containing poly[4-(1-adamantyl)styrene] hard segments". en. In: *Macromolecules* 41.14 (July 2008), pp. 5502–5508.

[58] Helmut Muenstedt, Nikolaos Katsikis, and Joachim Kaschta. "Rheological properties of poly(methyl methacrylate)/nanoclay composites as investigated by creep recovery in shear". en. In: *Macromolecules* 41.24 (Dec. 2008), pp. 9777–9783.

[59] Swati Singla and Haskell W Beckham. "Miscible blends of cyclic poly(oxyethylene) in linear polystyrene". en. In: *Macromolecules* 41.24 (Dec. 2008), pp. 9784–9792.

[60] Raquel Rodrıguez et al. "Correlation of silicone incorporation into hybrid acrylic coatings with the resulting hydrophobic and thermal properties". en. In: *Macromolecules* 41.22 (Nov. 2008), pp. 8537–8546.

[61] Cécile M Gibon et al. "Control of morphology and crystallization in polyelectrolyte/polymer blends". en. In: *Macromolecules* 41.15 (Aug. 2008), pp. 5744–5752.

[62] Erik B Berda and Kenneth B Wagener. "Inducing pendant group interactions in precision polyolefins: Synthesis and thermal behavior". en. In: *Macromolecules* 41.14 (July 2008), pp. 5116–5122.

[63] Nan Cheng et al. "Thickness-dependent properties of polyzwitterionic brushes". en. In: *Macromolecules* 41.17 (Sept. 2008), pp. 6317–6321.

[64] Sven Fleischmann, Hartmut Komber, and Brigitte Voit. "Diblock copolymers as scaffolds for efficient functionalization via click chemistry". en. In: *Macromolecules* 41.14 (July 2008), pp. 5255–5264.

[65] Michelle M Mok et al. “Microphase separation and shear alignment of gradient copolymers: Melt rheology and small-angle X-ray scattering analysis”. en. In: *Macromolecules* 41.15 (Aug. 2008), pp. 5818–5829.

[66] J A Pathak et al. “Structure evolution in a polyurea segmented block copolymer because of mechanical deformation”. en. In: *Macromolecules* 41.20 (Oct. 2008), pp. 7543–7548.

[67] Jana Herzberger and Holger Frey. “Epicyanohydrin: Polymerization by monomer activation gives access to nitrile-, amino-, and carboxyl-functional poly(ethylene glycol)”. en. In: *Macromolecules* 48.22 (Nov. 2015), pp. 8144–8153.

[68] Wolfgang H Binder et al. “Homologous poly(isobutylene)s: Poly(isobutylene)/high-density poly(ethylene) hybrid polymers”. en. In: *Macromolecules* 41.22 (Nov. 2008), pp. 8405–8412.

[69] Hai-Mu Ye et al. “Left- or right-handed lamellar twists in poly[(R)-3-hydroxyvalerate] banded spherulite: Dependence on growth axis”. en. In: *Macromolecules* 42.3 (Feb. 2009), pp. 694–701.

[70] E Bhoje Gowd, Kohji Tashiro, and C Ramesh. “Role of solvent molecules as a trigger for the crystal phase transition of syndiotactic polystyrene/solvent complex”. en. In: *Macromolecules* 41.24 (Dec. 2008), pp. 9814–9818.

[71] Florian J Stadler et al. “Linear viscoelastic rheology of moderately entangled telechelic polybutadiene temporary networks”. en. In: *Macromolecules* 42.16 (Aug. 2009), pp. 6181–6192.

[72] Nahrain E Kamber et al. “N-heterocyclic carbenes for the organocatalytic ring-opening polymerization of ϵ-caprolactone”. en. In: *Macromolecules* 42.5 (Mar. 2009), pp. 1634–1639.

[73] Guoqing Zhang et al. “Difluoroboron dibenzoylmethane PCL-PLA block copolymers: Matrix effects on room temperature phosphorescence”. en. In: *Macromolecules* 42.8 (Apr. 2009), pp. 3162–3169.

[74] Jung Min Lee et al. “One-step synthetic route for conducting Core-Shell poly(styrene/pyrrole) nanoparticles”. en. In: *Macromolecules* 42.13 (July 2009), pp. 4511–4519.

[75] Qian Zhang and C Geraldine Bazuin. “Liquid crystallinity and other properties in complexes of cationic azo-containing surfactomesogens with poly(styrenesulfonate)”. en. In: *Macromolecules* 42.13 (July 2009), pp. 4775–4786.

[76] Åsa M Ronkvist et al. “Cutinase-catalyzed hydrolysis of poly(ethylene terephthalate)”. en. In: *Macromolecules* 42.14 (July 2009), pp. 5128–5138.

[77] Chiara Neto, Michael James, and Andrew M Telford. “On the composition of the top layer of microphase separated thin PS-PEO films”. en. In: *Macromolecules* 42.13 (July 2009), pp. 4801–4808.

[78] Omar Al-Khayat et al. “Chain collapse and interfacial slip of polystyrene films in good/nonsolvent vapor mixtures”. en. In: *Macromolecules* 49.4 (Feb. 2016), pp. 1344–1352.

[79] Saki Tamura et al. “Thermosensitive self-assembly of diblock copolymers with lower critical micellization temperatures in an ionic liquid”. en. In: *Macromolecules* 42.16 (Aug. 2009), pp. 6239–6244.

[80] Matthieu F Dumont et al. “Synthesis and characterization of functionalized polysiloxane for the stabilization of catalytically active metal nanoparticles”. en. In: *Macromolecules* 42.14 (July 2009), pp. 4937–4940.

[81] Jessica N Hoskins and Scott M Grayson. “Synthesis and degradation behavior of cyclic poly(ϵ-caprolactone)”. en. In: *Macromolecules* 42.17 (Sept. 2009), pp. 6406–6413.

[82] Veronique Lachat et al. “Molecular origin of solvent resistance of polyacrylonitrile”. en. In: *Macromolecules* 42.18 (Sept. 2009), pp. 7103–7107.

[83] Wen-Hsiang Chen et al. “N,N,N’,N’- tetraphenyl-1,4-phenylenediamine-Fluorene alternating conjugated polymer: Synthesis, characterization, and electrochromic application”. en. In: *Macromolecules* 43.5 (Mar. 2010), pp. 2236–2243.

[84] Avanish Bharati et al. “Effect of compatibilization on interfacial polarization and intrinsic length scales in biphasic polymer blends of PαMSAN and PMMA: A combined experimental and modeling dielectric study”. en. In: *Macromolecules* 49.4 (Feb. 2016), pp. 1464–1478.

[85] Markus Sitzmann. *CACTUS Chemical Identifier Resolver.* `https://cactus.nci.nih.gov/chemical/structure`. Online; accessed 2022.10.10.

[86] Daniel M. Lowe et al. "Chemical Name to Structure: OPSIN, an Open Source Solution". In: *Journal of Chemical Information and Modeling* 51.3 (2011). PMID: 21384929, pp. 739–753. DOI: `10.1021/ci100384d`. eprint: `https://doi.org/10.1021/ci100384d`. URL: `https://doi.org/10.1021/ci100384d`.

[87] *RDKit: Open-source cheminformatics.* `http://www.rdkit.org`.

[88] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks.* 2016. DOI: `10.48550/ARXIV.1609.02907`. URL: `https://arxiv.org/abs/1609.02907`.

[89] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry.* 2017. DOI: `10.48550/ARXIV.1704.01212`. URL: `https://arxiv.org/abs/1704.01212`.

[90] Benjamin Sanchez-Lengeling et al. "Evaluating Attribution for Graph Neural Networks". In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 5898–5910. URL: `https://proceedings.neurips.cc/paper/2020/file/417fbbf2e9d5a28a855a11894b2e795a-Paper.pdf`.

[91] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[92] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds.* 2019.

[93] Yufengwhy. *PyGCN implementation without pitfalls.* `https://github.com/yufengwhy/pygcn`. Online; accessed 2022.10.10.

Scalability of Targeted Adaptive Design on SambaNova

PI: Marieme Ngom, Co-I: Noah H. Paulson

October 2022

1 Introduction to the Science problem

Atomic Layer Deposition (ALD) is an advanced technique used for depositing thin films with a range of applications including semiconductors, energy storage systems, and, biomedical devices. In this project, we consider ALD reactions that involve two chemicals, precursor A and precursor B, that react cyclically with the surface of a substrate in a sequential, non-overlapping manner. Each cycle is composed of four sequences:

1. A dose period t_1 during which the substrate is exposed to precursor A,
2. A purge period t_2 during which there is no precursor exposure,
3. A dose period t_3 during which the substrate is exposed to precursor B,
4. A final purge period t_4 during which there is no precursor exposure

At the end of each cycle, the Growth Per Cycle (GPC) which corresponds the net mass or thickness variation from the preceding ALD cycle is evaluated. The dose and purge times (t_1, t_2, t_3, t_4) can be controlled to ensure saturation of the GPC in minimum time. Furthermore, a set of other parameters relative to the properties of the precursors and the substrate can also be controlled to steer the saturation of the GPC. These include the temperature T in Kelvin (K), the site area A in square meters (m^2), pressures (p_A, p_B) in Pascal (Pa), the molecular masses (M_A, M_B) in atomic mass units, the sticking probabilities (β_A, β_B), the characteristic times for precursor evacuation (t_{p_A}, t_{p_B}) in seconds (s) and the mass changes (dm_A, dm_B) for that particular half-cycle in nanograms per square centimeters (ng/cm^2). In [3], the authors proposed three

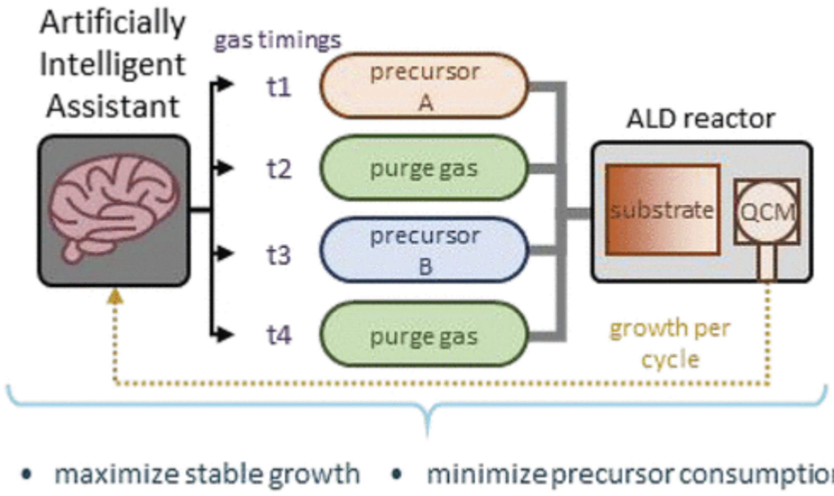


Figure 1: Paulson et al., ACS Appl. Mater. Interfaces 2021, 13, 17022-17033

optimization strategies (Random Optimization (RO), Bayesian Optimization (BO), and Expert Systems Optimization (ESO)) to determine dose and purge times that would yield optimal GPC saturation for four ALD processes simulated using a physics-based model surrogate. In this work, we fix the dose/purge times as well as all the other precursors/substrate related properties except for the characteristic precursor evacuation times (t_{p_A}, t_{p_B}) and the mass changes (dm_A, dm_B) and use Targeted Adaptive Design (TAD) to locate optimal values for these four parameters that would yield a particular GPC curve.

2 Description of the AI model and implementation

Targeted Adaptive Design (TAD) [2] is a novel probabilistic machine learning algorithm that aims at autonomously and efficiently locating optimal control parameters that would yield a desired target output. The motivation behind TAD stems from additive manufacturing and material synthesis where scientists and engineers are often faced with the problem of locating control settings that would result in a target design f_T with specified tolerance while accounting for measurement and model uncertainty. Furthermore, as for the ALD process, the control settings space (dose/purge times, precursors properties), as well as the design feature space (stable growth, precursor consumption), are usually high-dimensional. In addition, the mapping f between the control space and the design space is only ascertainable through noisy measurements

$$g_k = f(x) + \epsilon_k,\ k = 1, \dots, N$$

where $\boldsymbol{x}_k \equiv \{x_{k,j} \in X : j = 1, \dots, N_k\}$ and ϵ_k is a Gaussian noise term, and sometimes expensive simulations. TAD starts by first placing a Gaussian Process (GP) model on the unknown mapping and then by optimizing an acquisition function to not only determine the next candidate control settings but also the best region to look that candidate in. This allows for efficient lookup of the control space. Thereby TAD is a batch iterative method, at each stage we have:

- A set $\boldsymbol{x}_1 \equiv \{x_{1,k} \in X : k = 1, \dots, N_1\}$ of previously acquired samples with corresponding data $\boldsymbol{g}_1 \equiv f(\boldsymbol{x}_1) + \boldsymbol{\epsilon}_1$,
- A set $\boldsymbol{x}_2 \equiv \{x_{2,k} \in X : k = 1, \dots, N_2\}$ representing a batch of samples to be acquired at the current iteration with corresponding data $\boldsymbol{g}_2 \equiv f(\boldsymbol{x}_2) + \boldsymbol{\epsilon}_2$,
- A target point x representing a candidate problem solution with the corresponding latent noise-free target value $f(x)$ to be estimated.

In this case, he GP surrogate function produces the joint normal distribution

$$\begin{bmatrix} f(x) \\ \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \end{bmatrix} \sim \mathcal{N} \left\{ \begin{bmatrix} \mu(x) \\ \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} K_{xx} & \boldsymbol{K}_{x1} & \boldsymbol{K}_{x2} \\ \boldsymbol{K}_{1x} & \boldsymbol{K}_{11} + \boldsymbol{\Sigma}_1 & \boldsymbol{K}_{12} \\ \boldsymbol{K}_{2x} & \boldsymbol{K}_{21} & \boldsymbol{K}_{22} + \boldsymbol{\Sigma}_2 \end{bmatrix} \right\}.$$

From this, we can obtain the obtain the conditionals

$$\begin{aligned} f(x)|(\boldsymbol{g}_1, \boldsymbol{g}_2) &\sim \mathcal{N}\left\{ p^{(f(x)|1+2)}, Q^{(f(x)|1+2)} \right\} \\ \boldsymbol{g}_2|\boldsymbol{g}_1 &\sim \mathcal{N}\left\{ \boldsymbol{p}^{(2|1)}, \boldsymbol{Q}^{(2|1)} \right\} \end{aligned}$$

where the quantities $p^{(f(x)|1+2)}$, $Q^{(f(x)|1+2)}$, $\boldsymbol{p}^{(2|1)}$, and $\boldsymbol{Q}^{(2|1)}$ are derived from the μ and K by the standard GP methods [5]. We then derive the log-posterior predictive debsitive if target value f_T at current target point x as

$$\begin{aligned} \mathcal{L}_P(x, \boldsymbol{x}_1, \boldsymbol{g}_1, \boldsymbol{x}_2, \boldsymbol{g}_2) = &\log \pi \left[f(x) = f_T \,|\, 1+2 \right] \\ &-\frac{1}{2} \log\det Q^{(f(x)|1+2)} \\ &-\frac{1}{2} \left(f_T - p^{(f(x)|1+2)} \right)^T \left(Q^{(f(x)|1+2)} \right)^{-1} \left(f_T - p^{(f(x)|1+2)} \right). \end{aligned}$$

up to a constant. Given that we don't have the data samples g_2 yet, we take the expectation of $\mathcal{L}_P$ over the distribution of $\boldsymbol{g}_2|\boldsymbol{g}_1$

$$\mathcal{L}_{TAD}(x, \boldsymbol{x}_1, \boldsymbol{g}_1, \boldsymbol{x}_2) \equiv E_{\boldsymbol{g}_2|\boldsymbol{g}_1} \left\{ \mathcal{L}_{\mathcal{P}}(x, \boldsymbol{x}_1, \boldsymbol{g}_1, \boldsymbol{x}_2, \boldsymbol{g}_2) \right\}.$$

which gives rise to TAD's acquisition function

$$\mathcal{L}_{TAD}(x,\boldsymbol{x}_1,\boldsymbol{g}_1,\boldsymbol{x}_2) = \underbrace{-\frac{1}{2}\log\det\left[Q^{(f(x)|1)}-T\right] - \overbrace{\frac{1}{2}\left(f_T - p^{(f(x)|1)}\right)^T\left(Q^{(f(x)|1)}-T\right)^{-1}\left(f_T - p^{(f(x)|1)}\right)}^{\text{datafit term}}}_{\text{log-Gaussian term, exploitation}}$$
$$\underbrace{-\frac{1}{2}\text{Trace}\left\{T\left(Q^{(f(x)|1)}-T\right)^{-1}\right\}}_{\text{Trace term, exploration}}$$

where

$$T = \left[\boldsymbol{K}_{x2} - \boldsymbol{K}_{x1}\left(\boldsymbol{K}_{11}+\boldsymbol{\Sigma}_1\right)^{-1}\boldsymbol{K}_{12}\right]\left(\boldsymbol{Q}^{(2|1)}\right)^{-1}\left[\boldsymbol{K}_{2x} - \boldsymbol{K}_{21}\left(\boldsymbol{K}_{11}+\boldsymbol{\Sigma}_1\right)^{-1}\boldsymbol{K}_{1x}\right].$$

contains all the dependence on the latent variable $\boldsymbol{x}_2$. The TAD acquisition function incorporates the well-known exploration/exploitation trade-off. Furthermore, it is continuously differentiable and easy to compute. A high-level summary of the TAD algorithm is provided in 1.

Algorithm 1 Targeted Adaptive Design

```
Input: $\boldsymbol{x}_1$, set of initial design points of size $N_1$, and the corresponding samples $\boldsymbol{g}_1$
Input: $\boldsymbol{x}_2$, initial cluster of proposed sample points of size $N_2$, and the corresponding samples $\boldsymbol{g}_2$
Input: $x$, a target control point initialization
Input: $Converged \leftarrow$ False, $Check\_Model \leftarrow$ False
Input: $iter \leftarrow 0$
 1: while not $Converged$ do
 2:     $iter \leftarrow iter + 1$
 3:     Fit a GP surrogate to $f(\boldsymbol{x}_1)$
 4:     $x^{(0)} \leftarrow x$, $\boldsymbol{x}_2^{(0)} \leftarrow \boldsymbol{x}_2$
 5:     $\boldsymbol{x}, \boldsymbol{x}_2 \leftarrow \arg\max_{x,\boldsymbol{x}_2} \mathcal{L}_{TAD}(x,\boldsymbol{x}_1,\boldsymbol{g}_1,\boldsymbol{x}_2)$
 6:     Call Procedure CHECK_CONVERGENCE
 7:     if not $Converged$ then
 8:         Acquire data $\boldsymbol{g}_2$ at $\boldsymbol{x}_2$ and $g$ at $x$
 9:         $\boldsymbol{x}_1 \leftarrow \boldsymbol{x}_1 \cup \boldsymbol{x}_2 \cup x$, $\boldsymbol{g}_1 \leftarrow \boldsymbol{g}_1 \cup \boldsymbol{g}_2 \cup g$
10:     end if
11: end while
```

TAD algorithm encompasses much more technical details such as a dynamical model validation as well as convergence testing procedures, however, to keep this report self-contained we omit them and refer readers to [2].

One widely-known bottleneck of GPs is the scalability limitations due to the numerical inversion of a $EN \times EN$ covariance matrix where N is the data size and E the dimension of the design space. The size of the covariance matrix is thus controlled by the size of the data but also the size of the design space and can become large quickly. If using Cholesky to perform the factorization of the covariance matrix, the scaling is $O(E^3N^3)$ which is prohibitive. To circumvent this, we implemented TAD using Gpytorch [4], a software built on top of Pytorch that is particularly adapted for large datasets. Gpytorch incorporates fast structured kernel methods such as KISS-GP [6] which scales nearly linearly. In addition, GPytorch incorporates the LanczOs Variance of Estimate (LOVE) [1] which can compute predictive variances in constant time.

3 What was needed to get the model running on the AI Accelerator

As mentioned above, each TAD iteration can be broken into two steps: the GP fitting step and the acquisition one. In TAD, the GP fitting is performed using the ExactGP model from GPytorch which cannot be used in conjunction with stochastic optimization. Therefore, the loss criterion could not be called from *init* and the loss function had to be computed outside of the forward call of the model. Furthermore, GPytorch, although built on top of Pytorch, has many operations that are not supported on SambaNova such as their efficient routines for Matrix-Vector Multiplications (MVM) and the implementation of their BlackBox Matrix-Matrix Multiplication method (BBMM). We believe focus should be put on porting the corresponding gpytorch functions (located in $gpytorch/functions$) on SambaNova.

The size and the scope of incorporating Gpytorch to SambaNova were such that it could not be completed in time for the summer expedition. .

4 Performance Evaluation

TAD+ALD on ThetaGPU Our goal is to reproduce a GPC curve using the ALD physics-model proposed in [**?**]. To that aim we fix all the controllable parameters mentioned in 1 to the following values:

$$t_1 = t_2 = t_3 = t_4 = 0.25s,\ T = 473K,\ A = 22.5e{-}20m^2,\ p_A = 26.66Pa = p_B,\ M_A = 72AMU,\ M_B = 18AMU$$

$$\beta_A = 1e-3,\ \beta_B = 1e-4,\ t_{p_A} = 0.2s,\ t_{p_B} = 0.3s,\ d_{m_A} = 0.8ng/cm^2,\ d_{m_B} = 0.5ng/cm^2$$

and record the corresponding growth curve as our target design f_T. The size of f_T is controlled by the number E that incidentally controls the size of the covariance matrices at play in TAD. We show results for $E = 50$ points corresponding to 50 full ALD cycle. We start with an initial x_1 sample of size 20 and add 10 x_2 points plus 1 target point at each iteration. After one iteration, we have acquired 31 points resulting in a covariance matrix of size 550. This means that using traditional Cholesky method would become rapidly as the number of TAD iterations grow. In addition, we set the tolerance to 10% meaning that we are looking for control parameters that would yield the target growth rate within ±10%. TAD converged to the solution in 2 iterations spendng approximately 4.3s on the GP fitting step and 15.8s in the optimization step.

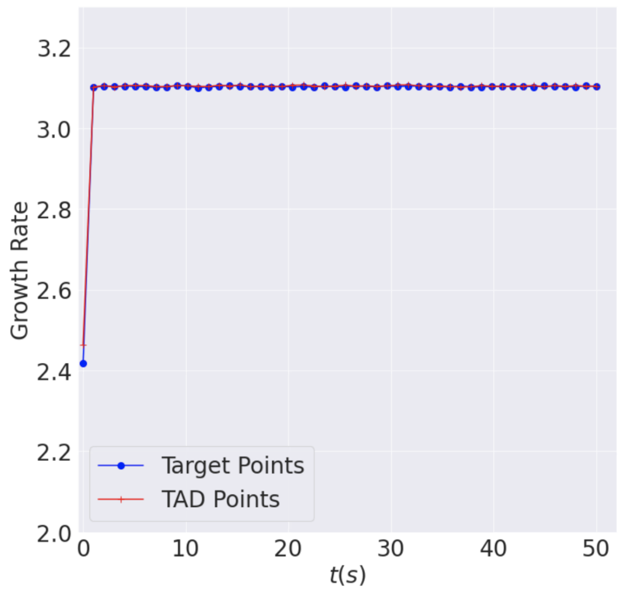


Figure 2: Paulson et al., ACS Appl. Mater. Interfaces 2021, 13, 17022-17033

The target and TAD growth rate curves are shown in Figure 2

5 Conclusion and next steps

In the recent years. tremendous advances were made to solve the scalability bottleneck of Gaussian Processes making them one of the most powerful and reliable regression tools available. GPytorch, in particular is a highly efficient and modular implementation of GPs built on top of pytorch. Hence, it is an important methodology to have available on SambaNova. Although we were not able to have GPytorch fully supported on SambaNova during the length of this project, implementing it has been added to SambaNova's roadmap. This LDRD was the occasion to apply our newly developed method TAD to an important scientific process, ALD. We successfully ran tests on ThetaGPU and look forward to continuing the collaboration with the SambaNova team to have GPytorch supported on their system.

6 Acknowledgements

We would like to thank Murali Emani and Varuni Sastry from ALCF, and Rick Weisner from SambaNova for their valuable comments and support during this project. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[2] C. Graziani and M. Ngom. Targeted adaptive design. `https://arxiv.org/pdf/2205.14208.pdf`, 2022.

[3] Paulson N.H., Yanguas-Gil A.and Abuomar O.Y., and Elam J.W. Intelligent agents for the optimization of atomic layer deposition. *ACS Applied Materials and Interfaces*, 13(4):17022–17033, 2021.

[4] G. Pleiss, J. R. Gardner, K. Q. Weinberger, A. G. Wilson, and M. Balandat. Gpytorch. `https://github.com/cornellius-gp/gpytorch`, 2022.

[5] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning.* Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

[6] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784. PMLR, 2015.

Ensemble forward gradients on AI hardware

Jan Hückelheim (PI), Krishnan Raghavan, Pinaki Pal, Tadbhagya Kumar

September 2022

Abstract

Computational Fluid Dynamics (CFD) is used in the design and optimization of gas turbines and many other industrial and scientific applications. Its practical use is often limited by the high computational cost, and the requirement of accurately resolving near-wall flow is a significant contributor to this cost. Machine learning and other data-driven methods can complement existing wall models. Nevertheless, training accurate and therefore often large models presents a large cost in itself, in part due to the computational effort and memory footprint required by back-propagation. Recent work has presented alternatives for computing gradients of neural networks that does not require a separate forward and backward sweep, nor requires storage of intermediate results between sweeps. The approach instead computes an unbiased estimator for the gradient in a single forward sweep. We implemented this approach in a framework built on top of PyTorch. In this paper we discuss the application of this approach to a CFD wall model that could potentially be used as a surrogate in wall-bounded flow simulations to reduce the computational overhead while preserving predictive accuracy.

1 Introduction

Reverse mode automatic differentiation and back-propagation can efficiently compute gradients, but often require large amounts of memory due to the need to store intermediate state information, and are not supported on all hardware, particularly on some AI accelerators. In this work, we explore the use of forward gradients as an alternative to back-propagation in the context of data-driven machine learning models that are used to augment conventional computational fluid dynamics (CFD) methods for simulating film cooling of turbo-machinery components such as gas turbines.

To perform parametric design studies of such components, it is often desired to have high-fidelity simulations that capture a wide range of spatio-temporal scales. Wall resolved large-eddy simulations (WRLES) [1, 2, 3, 4, 5] capture the unsteady features and provide a feasible way for simulating complex turbulent flows at a reasonable cost. However, the computational cost incurred due to very high resolution needed to resolve the viscous scale near the wall is still high. To overcome this, wall models are used to avoid the need to resolve the near-wall region, providing a feasible way for LES of wall-bounded flows at high Reynolds numbers.

Recent advancements in machine learning and high performance computing have motivated new efforts to develop data-driven methods to complement existing wall models. Some of the popular methods and architectures in the literature are the Random Forest regression ([6, 7]), artificial neural networks [8, 9, 10, 11], and convolutional neural networks[12, 13, 14]. In most of these approaches, high fidelity simulations are performed (WRLES) and data is gathered for flow features (velocity of the fluid, pressure gradients, temperature). A data-driven regression model that predicts the wall-shear stress is trained on these flow features, and then used as a wall model for LES of turbulent channel flows.

2 Application

A data-driven wall model was developed to predict shear stress in a wall-modeled large-eddy simulation (LES) of gas turbine film cooling. Film cooling is a popular technique used to reduce turbine temperature and protect them from thermal failure. As the hot gases exit from the combustor and enter the turbine stage, the temperatures can be higher than the melting point of turbine stage materials. Cooling strategies thus become necessary for thermal management of the turbine blades. Since conventional CFD models suffer from the aforementioned limitations, deep neural network-based wall models are being investigated as a surrogate in wall-bounded flow simulations to reduce the computational overhead while preserving predictive accuracy.

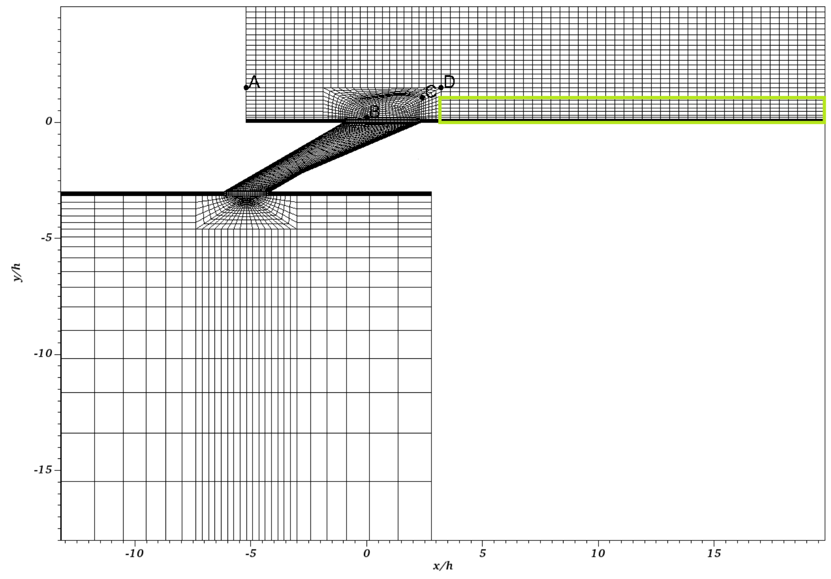


Figure 1: Wall resolved LES mesh of the film cooling setup. The highlighted region (green) depicts the domain of data collection for training the ML wall model

2.1 Data Generation

The data used in this work to train data-driven model is taken from the wall-resolved(WR) LES of a 7-7-7 cooling hole configuration detailed in [5] for two blowing ratios (BR1): 1 and 1.5. It consists of three parts: a plenum feeding cool air to a flat surface via a single row of cooling holes (represented with a single hole with periodic boundary conditions in the spanwise direction). The flow was simulated using the Nek5000 [15] platform, a higher-order spectral element CFD code, using its Low Mach flow solver. Figure 1 depicts the wall resolved LES mesh used to perform the CFD simulation, and the green box highlights the flow data domain which is used to train the data-driven model. The flow data in the green highlighted region was collected at 91 planes for $y \in [0.05, 0.5]$ such that $z \in [-3, 3]$ and $x \in [3.2, 19.6]$.

2.2 Data-driven Modeling

Fluid velocity and velocity gradients, pressure gradients, and the density of the fluid are extracted from the WRLES simulations and serve as the base features, whereas the shear stress is used as the output. These flow features are combined from multiple neighboring x,y,z locations in a stencil-like manner and fed as the input to the neural network. The network consists of an input layer of size equal to the number of features used and 3 hidden layers with 64 neurons each with a dropout value of 0.1 and ReLU activation function. The output layer is a dense linear layer of size one and predicts the wall shear stress in the streamwise direction (τ_{xy}). The prepared input - output data is scaled in the range [-1,1] using Min-Max scaling:

$$x^* = -1 + 2\frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

where x is the variable of interest. The dataset for BR1 is randomly shuffled and an 80%-20% split is performed to generate training and validation datasets respectively. The loss function used to train the network is the mean squared error (E_w):

$$E_w = \frac{1}{N}\Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

The neural network model is implemented in PyTorch and is trained using the Adam optimizer with early stopping criterion. The trained model performance is assessed using the Pearson's correlation coefficient of the predicted wall shear stress against the WRLES data on our validation set.

3 Forward Gradients

The key contribution of this paper is the implementation and use of forward gradients for a physical science use-case, not demonstrated earlier. We summarize the theory as shown in [16], followed by a description of our implementation strategy.

3.1 Theory

Given a objective function $f(\theta) : \mathbb{R}^n \to \mathbb{R}^n$, we seek to solve the optimization problem

$$\underset{\theta \in \mathbf{\Omega}}{min} f(\theta). \tag{3}$$

Intuitively, we seek to find a θ in the search space $\mathbf{\Omega}$ such that the objective function value is minimized. The objective function can take many forms but is usually expected to be twice differentiable. This is required because, typically, the process of finding the minima involves a search using the gradients of the objective function denoted $\nabla_\theta f(\theta)$. Since exact calculation of $\nabla_\theta f(\theta)$ is usually expensive, several simplifications are performed based on the assumption that, what is usually required for efficient learning is the directional derivative of f with respect to θ. This can be accomplished by forward gradients [16] $g(\theta)$ such that

$$g(\theta) = (\nabla_\theta f(\theta).\mathbf{v})\mathbf{v} \tag{4}$$

where $\mathbf{v}$ is a perturbation vector chosen as a multivariate random variable sampled $\mathbf{v} \sim p(\mathbf{v})$ such that the scalar components v_i are independent with zero mean and unit variance for all i's. The key insight here is that, $g(\theta)$ through the perturbation vector allows us to estimate the directional derivatives without needing to explicitly compute the gradient $\nabla_\theta f(\theta)$. This allows for a runtime advantage over typical back-propagation where we can interpret the directions of update without needing to distinguish the updates for each scalar parameter in θ. The process of evaluating forward gradients typically involves a three step process where we

1. sample a random perturbation $\mathbf{v} \sim p(\mathbf{v})$, which has the same size with f's argument,
2. run forward mode autodiff to evaluate $(\nabla_\theta f(\theta).\mathbf{v})$ and $f(\theta)$ simultaneously, and finally
3. multiply $(\nabla_\theta f(\theta).\mathbf{v})$ with the perturbation vector $\mathbf{v}$ to obtain the forward gradients $g(\theta)$.

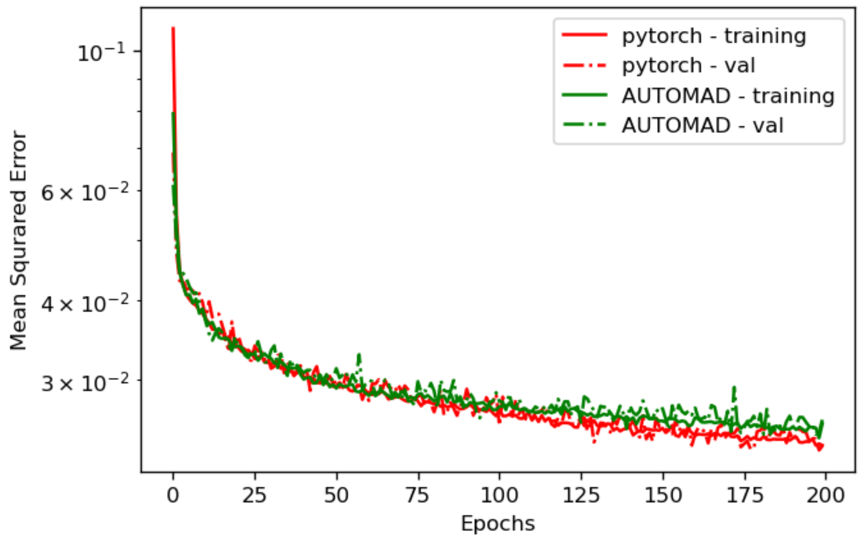
As long as $g(\theta)$ is similar to the true gradients, or points in a descent direction, the optimization problem can still be solved efficiently. It is shown in [16] that the forward gradients are an unbiased estimator of $\nabla_\theta f(\theta)$ and can be used to solve the optimization problem (4).

3.2 Implementation

Forward gradients are propagated through the model using the forward mode, also known as tangent mode, of automatic differentiation [17]. Since the code used by [16] is not yet publicly available at the time of writing, we used our own framework, which not only supports the computation of randomized forward gradients, but also computing entire Jacobians of the network or individual layers, as well as mixed-mode differentiation that allows a user to freely combine forward gradients for some layers with conventional back-propagation for other layers. The implementation is already freely available on github[1].

The framework implements a so-called *tangent model* for selected PyTorch layers, which implement the forward mode of automatic differentiation for these layers. It also provides a simple way to combine multiple layers in a way that mimics standard PyTorch usage, and can therefore be used as a drop-in replacement for PyTorch in many cases. The tangent model of individual layers is implemented by using existing PyTorch layers underneath, which allows us to support GPUs, AI accelerators, and other hardware platforms efficiently. Since our framework can compute forward gradients without using PyTorch back-propagation, it can even be used to obtain approximate gradients for training on AI accelerators that do not officially support training due to their lack of back-propagation support.

[1]URL removed for anonymization



(a)

Figure 2: Comparison of loss evolution between the forward gradient approach and standard back-propagation for the wall-resolved LES dataset.

We discuss here the tangent model of a 2D convolution layer to illustrate the method. For each output pixel and output channel N_i and C_{out_j}, `Conv2d` computes an output value o by adding a bias to a weighted sum of cross correlations over the input values u at nearby pixels across input channels as

$$o(N_i, C_{out_j}) = b(C_{out_j}) + \sum_{k=0}^{C_{in}-1} w(C_{out_j}, k) \star u(N_i, k). \tag{5}$$

Differentiating with respect to the bias, weights and inputs yields

$$\dot{o}(N_i, C_{out_j}) = \dot{b}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \dot{w}(C_{out_j}, k) \star u(N_i, k) + \sum_{k=0}^{C_{in}-1} w(C_{out_j}, k) \star \dot{u}(N_i, k), \tag{6}$$

where derivative terms are denoted using a dot above. To compute forward gradients, the term $\dot{w}$ needs to be replaced with a random vector as discussed in Section 3.1. The term $\dot{u}$ needs to be replaced with the output of the preceding tangent layer, or dropped in case of the first layer of the network. The tangent model shown in (6) can itself be computed using two calls to `Conv2d`.

4 Performance Evaluation

Figure 2a shows the loss function evolution for both the forward gradient approach and back-propagation approach. The network is trained using ADAM optimizer, with an initial learning rate $\eta = 2 \times 10^{-4}$. As training progresses, the learning rate is decayed by a factor $\gamma = 0.5$ if the validation loss does not change for 20 epochs. It can be seen that the mean squared error decreases for the training and validation set, both for conventional back-propagation and forward gradients. The decrease happens at comparable speed as measured in number of epochs.

The main selling point of forward gradients is probably that it can run even on platforms that do not support back-propagation, and our implementation demonstrates that training is indeed possible without resorting to PyTorch's built-in back-propagation, without storing intermediate results at each layer step, and without reversal of the computation or data flow.

Even though the forward gradient approach uses approximately the same number of flops as back-propagation, our implementation is ca. 10x slower than back-propagation on a CPU. We plan to further investigate (and hopefully remove) the reason for this performance degradation before performing a full benchmark on AI hardware.

5 Conclusion and next steps

Forward gradients seem to be effective for training in this application. In future work, we plan to investigate different optimizer settings, compute the mean from multiple forward gradient computations with varying random seeds, which is likely to yield a more accurate representation of the true gradient, and further improve the performance of our implementation on CPU, GPU, and AI platforms.

6 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] Kunlun Liu and Richard Pletcher. Large eddy simulation of discrete-hole film cooling in a flat plate turbulent boundary layer. In *38th AIAA Thermophysics Conference*, page 4944, 2005.

[2] X Guo, W Schröder, and M Meinke. Large-eddy simulations of film cooling flows. *Computers & Fluids*, 35(6):587–606, 2006.

[3] Yulia V Peet and Sanjiva K Lele. Near field of film cooling jet issued into a flat plate boundary layer: Les study. In *Turbo Expo: Power for Land, Sea, and Air*, volume 43147, pages 409–418, 2008.

[4] Todd A Oliver, Joshua B Anderson, David G Bogard, Robert D Moser, and Gregory Laskowski. Implicit les for shaped-hole film cooling flow. In *Turbo Expo: Power for Land, Sea, and Air*, volume 50879, page V05AT12A005. American Society of Mechanical Engineers, 2017.

[5] Austin C Nunno, Sicong Wu, Muhsin Ameen, Pinaki Pal, Prithwish Kundu, Ahmed Abouhussein, Yulia Peet, Michael Joly, and Peter Cocks. Wall-resolved les study of shaped-hole film cooling flow for varying hole orientation. In *AIAA SCITECH 2022 Forum*, page 1404, 2022.

[6] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, 2(3):034603, 2017.

[7] Tadbhagya Kumar, Pinaki Pal, Austin C Nunno, Sicong Wu, Opeluwa Owoyele, Michael Joly, and Dima Tretiak. Development of a data-driven wall model for large-eddy simulation of gas turbine film cooling flows. In *AIAA SciTech Forum and Exposition*, 2023 (accepted).

[8] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[9] Zhideng Zhou, Guowei He, and Xiaolei Yang. Wall model based on neural networks for les of turbulent flows over periodic hills. *Physical Review Fluids*, 6(5):054610, 2021.

[10] XIA Yang, Suhaib Zafar, J-X Wang, and Heng Xiao. Predictive large-eddy-simulation wall modeling via physics-informed neural networks. *Physical Review Fluids*, 4(3):034602, 2019.

[11] Xinyi LD Huang, Xiang IA Yang, and Robert F Kunz. Wall-modeled large-eddy simulations of spanwise rotating turbulent channels—comparing a physics-based approach and a data-based approach. *Physics of Fluids*, 31(12):125105, 2019.

[12] Junhyuk Kim and Changhoon Lee. Prediction of turbulent heat transfer using convolutional neural networks. *Journal of Fluid Mechanics*, 882, 2020.

[13] Luca Guastoni, Miguel P Encinar, Philipp Schlatter, Hossein Azizpour, and Ricardo Vinuesa. Prediction of wall-bounded turbulence from wall quantities using convolutional neural networks. In *Journal of Physics: Conference Series*, volume 1522, page 012022. IOP Publishing, 2020.

[14] Naoki Moriya, Kai Fukami, Yusuke Nabae, Masaki Morimoto, Taichi Nakamura, and Koji Fukagata. Inserting machine-learned virtual wall velocity for large-eddy simulation of turbulent channel flows, 2021.

[15] Paul Fischer, James Lottes, Stefan Kerkemeier, Oana Marin, Katherine Heisey, Aleks Obabko, Elia Merzari, and Yulia Peet. Nek5000: User's manual. *Argonne National Laboratory, Lemont, IL, Technical Report No. ANL/MCS-TM-351*, 2015.

[16] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

[17] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

A pathway to OpenMP In the GraphCore Architecture

Jose M Monsalve Diaz, Esteban Rangel, Siddhisanket Raskar, Johannes Doerfert
Argonne National Laboratory, Lemont IL

October 15, 2022

Abstract

AI and ML have extended the limits of what computers and applications can do. However, as new architectures and accelerators are introduced, the number of software libraries, tools and systems that users must learn to use increases exponentially. System designers often provide support for commonly used AI/ML frameworks (e.g. Tensorflow and Pytorch), but these force applications to rely on Python. In the meantime, low level API and languages are also available, but they have a steep learning curve that makes it difficult for applications to use these systems. Furthermore, the tools and compilers used are often proprietary, limiting the ability to extend and improve these. On the other hand, scientific computation is often found in traditional programming languages like C/C++ and Fortran, and parallelization of scientific programs is achieved via OpenMP. While it is unrealistic to imagine full support for OpenMP in AI/ML accelerators, it is possible to use the offloading abstraction, worksharing, and data management abstractions in OpenMP to interact with low level interfaces of these systems. This work explores potential connections between GraphCore Poplar library, and OpenMP, by exposing program flow control and data management interfaces to the OpenMP applications. Moving forward, this could open the door for GraphCore accelerators for Argonne (and DoE) science applications using OpenMP offloading, like QMCPACK and OpenMC

1 Introduction

This document summarizes the changes that are needed in order to support OpenMP offloading on the Graphcore architecture. We also provide a preliminary mapping from OpenMP semantics to Graphcore semantics. The motivation of this work comes from an expedition to understand the use of Graphcore architectures and alike into scientific computation. While most AI applications are already ported to Machine Learning frameworks such as Tensorflow and Pythorch, scientific applications still rely on traditional programming models such as OpenMP. Furthermore, if we envision a true heterogeneous system in the future that can take advantage of these new architectures, it is necessary to lower the bar for programming this system. This work is an effort towards that objective.

1.1 Graphcore IPU

The Graphcore IPU-M2000 system [8] is designed to accelerate and support scale-up and scale-out machine learning applications. The IPU-M2000 system 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 1472 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. A great description of the hardware, and its capabilities, can be found in [7].

This work uses IPU-M2000 system but it is also applicable for previous generation Colossus IPU as well as the latest BOW IPU [4].

1.2 Graphcore Poplar SDK

Poplar is a graph-programming framework for the Graphcore Intelligence Processing Unit (IPU). Graphcore Poplar SDK [6] provides a mature platform for ML development and deployment. It 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 across the processor cores and their tightly coupled memory. This is achieved by compiling the timing of the compute, sync, and exchange phases into an executable graph and taking advantage of the BSP synchronization. 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. PopLibs, PopART, and PopTorch are available as open-source in the Graphcore GitHub repo. The Poplar SDK also includes the PopVision visualization and analysis tools that provide performance monitoring for the IPU-M2000 system.

1.3 Graph Description in OpenMP Tasks

OpenMP supports tasking programming model. A task is created by using the `omp task` directive, and edges are defined by `depend` clauses. However, OpenMP task graph description is fundamentally different to that of the Graphcore architecture. OpenMP tasking model relies on dynamic dependencies that are discovered during based on the order of execution during runtime. Furthermore, compiler analysis is not possible because dependencies may be determined by the result of a previous task. On the other hand, GraphCore requires the construction of static graphs prior to the beginning of the execution of the program in the IPU system. Furthermore, Graphcore limits the graphs to BSP-like execution models, while a set of control instructions allow to have some dynamic behavior on the BSP execution phases.

Some prior work has used delayed scheduling of OpenMP tasks to bridge the gap between these two models [10]. However, this is only possible if the dynamic behaviors are not dependent on the result of the computation within another task. This work uses a different approach. It provides limitations to the task OpenMP task definition to discover the task graph during compilation time. This responds to the need to explicitly form the IPU program, create the vertex, define the tensors, and connect the components of the GraphCore program. These limitations only affect the code defined in the `target` offloading region.

2 Mapping OpenMP to GraphCore IPUs

Here we describe our proposal for using GraphCore's IPUs as accelerators within the OpenMP device execution model. A straightforward approach for this implementation is to determine a mapping between the OpenMP API for offloading computation and expressing parallelism to the Poplar library SDK. The OpenMP device model supports multiple accelerators/co-processors which also fits nicely into GraphCore's multiple IPU design.

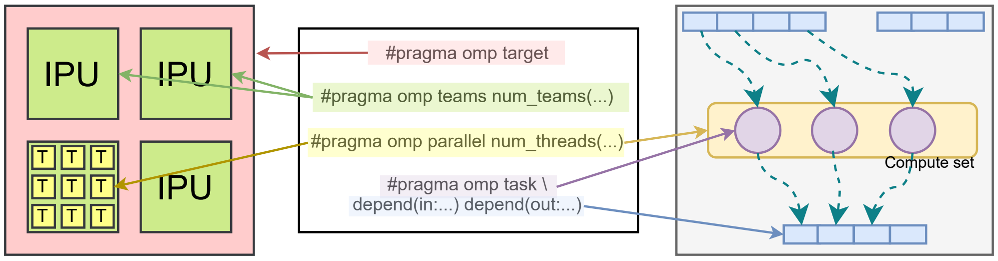

IPU
IPU
IPU
T T T
T T T
T T T
#pragma omp target
#pragma omp teams num_teams(...)
#pragma omp parallel num_threads(...)
#pragma omp task \
depend(in:...) depend(out:...)
Compute set

Figure 1: Syntax mapping between Graphcore architecture and OpenMP

2.1 IPU Offloading

The OpenMP `target` construct indicates where code should be offloaded onto a device. All code inside the target region is, effectively, translated to a Poplar `program` by the tool-chain described in section 3 Figure 2. Functionally, the Poplar program orchestrates copying data and executing compute sets using standard control flow. Listing 1 illustrates the OpenMP semantics to describe a Poplar program. In the example, we use the OpenMP `map` clause with the `alloc` map-type to create Poplar tensors for the mapped regions. These tensors are the inputs and outputs for the codelets described in subsection 2.2. The OpenMP `teams` construct creates a league of threads, with each thread starting it's own team of threads, i.e., contention group. We see this naturally mapping to the graph replication functionality of the Poplar SDK, where the same program is running in data-parallel fashion on (potentially) several sets of IPUs. Lastly, Poplar intrinsic functions, e.g., `copy`, can be explicitly called, or can be inferred, as in the use of the conditional statement.

Listing 1: OpenMP description of Poplar program

```
int v1[4]; // Static array sizes or mappings
float v2[4]; // Static array sizes or mappings
// Create tensors for mapped regions
#pragma omp target map(alloc: v1[:], v2[:])
  // Translated to graph replication
  #pragma omp teams num_teams(4)
  {
    const float c1[4] = {1.0,2.0,3.0,4.0};
    // This is an intrinsic funciton
    // that is part of the program.
    // Translated to prog.add(Copy(c1, v1));
    copy(c1,v1);
    // This is translated to an intrinsic
    // function that is part of the program.
    if (v1[0] > 3) { ...}
  }
```

Listing 2: OpenMP description of Poplar compute set and codelets

```
// Translated to a for loop of 4 elements that
// creates a compute set with 8 codelets, 4 of each task
#pragma omp parallel num_threads(4)
{ #pragma omp task depend(in:\
    v1[omp_get_thread_num():4-omp_get_thread_num()]) \
    depend(out: v2[omp_get_thread_num()])
  {
    // Body translated to codelet compute function.
    // Depend "in" and "out" do tensor mapping to the
    // inputs and outputs of the Codelets, based on tid
    *v2 = 0;
    for (const auto &v : v1)
      *v2 *= v;
  }
  #pragma omp task depend(in:\
    v1[omp_get_thread_num():4-omp_get_thread_num()]) \
    depend(out: v2[omp_get_thread_num()])
  {
    *v2 = 0;
    for (const auto &v : v1)
      *v2 += v;
  }
}
```

2.2 Expressing Parallelism

The Poplar library introduces the notion of a `codelet`, a small piece of code that runs on a single IPU tile and corresponds to a vertex in the graph abstraction. IPUs execute codelets concurrently in groups called *compute sets*, where each codelet is typically reading and writing a subset (slice) of the input/output tensors. Listing 2 illustrates using OpenMP semantics to express parallelism. The body of the OpenMP `task` region defines the codelet and is compiled by the back-end, `popc`, compiler. We then use the OpenMP `parallel` construct to express concurrency and use the `num_threads` clause to determine the size of the compute set. Note that the OpenMP semantics capture the MIMD execution capabilities of IPU architecture by allowing instantiation of multiple tasks, which in turn results in a compute set consisting of different codelets. Mapping of the compute set (as well as tensors) to the IPU tiles is linear and makes use of built-in functions, although more complex mappings could be investigated in future work.

2.3 Computational Graph and Data

The variables defined inside the OpenMP `depend` clause correspond to inputs and outputs [`in, out, inout`] of the task region. The input/output dependencies among vertices across compute sets define the computational graph, expressing the relationship between compute and data. Tensors are created either as allocated variables in the target region, or as mapped variables. If the mapping is `alloc`, the tensor is just added to the graph, however, if the mapping is [`to, from, tofrom`], this translates to data streams between CPU and IPU.

3 Implementation

We will proceed to describe the modifications that would be required in order to provide a full implementation of this prototype.

In order to implement OpenMP offloading there are roughly four elements that are required [1, 3, 9]: 1) a front end that can generate the appropriate OpenMP Runtime Calls and create the poplar program, 2) A compiler that can generate assembly code for the target architecture, 3) a new device plugin for the particular architecture, and 4) modifications to the Clang driver to cohesively coordinate the generation of code. This section describes how each of these components are design and implemented. Figure 2 shows the overall compiler infrastructure, highlighting the different components that need to be modified in order to support OpenMP offloading for the new GraphCore architecture.

Our implementation takes advantage of already existing software infrastructure for the GraphCore systems. The Poplar software allows programming the IPU systems by different mechanisms, from high level AI frameworks (e.g. Tensorflow and Pythorch) to low level C++ based languages. Two components of poplar are important for the description below. 1) The *popc* compiler, and 2) the *libpoplar* library and tools that is part of the Poplar SDK [5].

3.1 Backend compiler

The role of the back end compiler is to be able to generate assembly code for the target architecture. Creating a back-end from scratch is a challenging task, as it will require a lot of engineering time to provide

Listing 3: Example of a vertex in poplar

```
#include <poplar/Vertex.hpp>

class SumVertex : public poplar::Vertex {
public:
  poplar::Input<poplar::Vector<float>> in;
  poplar::Output<float> out;

  bool compute() {
    *out = 0;
    for (const auto &v : in) {
      *out += v;
    }
    return true;
  }
};
```

Listing 4: Code added by the popc frontend

```
extern "C" {
  __attribute__((target("worker")))
  __attribute__((colossus_vertex))
  int __runCodelet_SumVertex() {
    void *vertexPtr = __builtin_ipu_get_vertex_base();
    auto v = static_cast<SumVertex*>(vertexPtr);
    return v->compute();
}
```

a good quality backend. Fortunately, the GraphCore system software provides an LLVM compiler called *popc*. However, details on this compiler are limited, and it required additional steps to reverse engineer it.

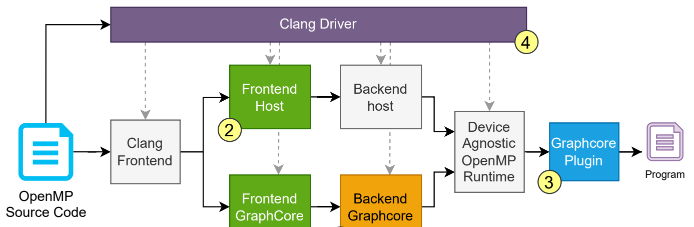


Figure 2: Implementation of OpenMP Support in LLVM. Modifications needed as they relate to the following subsections

Clang front-end generates LLVM-IR, however, the current documentation of the *popc* is really limited with respect to the LLVM-IR used by *popc*. By knowing that the compiler is LLVM based, we used some of the common debugging flags to understand the direct interaction with the backend of the *popc* compiler and directly send LLVM-IR to it. To achieve this, we ran a small application of a simple vertex [6] as seen in Listing 3. After building this code with *popc* using the `-###, -save-temps, -emit-llvm` and other debugging flags, it is possible to observe that this code will compile for three targets: *ipu1, ipu2,* and *cpu*. Additionally, the preprocessor will inserts the function in Listing 4 before passing it to the compilation pipeline. By reading the manual on low level assembly programming of IPUs, these functions are the bootstrapping mechanism used by IPU to execute a particular Codelet. Codelet is the name given to a vertex that is scheduled and executed in an IPU unit.

In addition, when comparing the LLVM-IR generated by *popc* and vanilla *clang* trunk compiler, there are no particular differences that would indicate incompatibilities between LLVM-IR generated with the original clang compiler and that generated by *popc*. Therefore, it should be possible to have the clang frontend generate the LLVM-IR, and use the *popc* compiler as back end. The driver will be in charge of making this connection, as it is done for other device offloading in OpenMP.

3.2 Frontend

OpenMP Offloading already supports outlining of target regions that are sent through different instantiation of the clang frontend [1] compiler for the same compilation unit, similar to that shown in Figure 2. Thanks to the use of a different target architecture, the logic in charge of generating LLVM-IR for OpenMP can change between devices. It is then possible to add a new device type that, when sent, will trigger a code generation that is specifically tailored for that type of devices. This works well for recognizing `omp tasks` regions within a `target region` as explained in section 2. It is possible to create code that assimilates that of Listing 3 and Listing 4, and generate an LLVM-IR that is sent to the *popc* compiler.

There is however, an additional challenge. Poplar requires the host to create a description of the poplar program and build the graph that connects the different Codelets, and tensors. In section 2 we already mention the appropriate syntax to do this connection. However the host front end will need to be heavily modified to generate an outlined function that uses the *poplib* API. An initial implementation that performs this operation will require considerable changes to the OpenMP front end, and will likely break the vendor-agnostic abstraction that the current Libomptarget uses [2]. This difficulties will be an interesting area of research for the future.

3.3 Plugin

Adding a new OpenMP pluging for a device is relatively simple. Libomptarget has been engineered to have a well define interface a device needs to support. The most important functions currently include:

`register_lib`, `unregister_lib`, `is_valid_binary`, `init_device`, `load_binary`, `data_{alloc, submit, retrieve, delete, exchange}`, and `run_{target_region,target_teams_region}` The difficulties, however, are that the poplar programming model is fundamentally different to these interfaces inherently inspired by GPUs. A first problem is that Poplar-like programs need a static description of the computation graph that is built in the host at runtime. To achieve this in little time, and implementation in the frontend may create new outlined regions that executes in the host prior to launching the kernel in the device. Nevertheless, the API calls of the current plugin consider an image to the device that is sent directly, and executed directly through the low level runtime APIs provided by the vendor. Poplar does not provide these mechanisms, and therefore, new modifications would need to be proposed. A second difficulty, is that functions such as `run_target_region` and alike do not have a use from the perspective of the graphcore architecture, as described in section 2. Perhaps, it would be possible to find use for the data mapping functions, but this is still an area of research.

3.4 Driver

The compiler driver is in charge of instantiating the different tools used during the compilation pipeline. In LLVM this is called the clang driver. Current implementation of OpenMP offloading already supports the creation of multiple compilation pipelines for different target architectures. By taking advantage of this feature, the clang driver has been modified to directly call the *popc* compiler [**?**]. This implementation still does not support an appropriate front end, or directly generates the needed poplar calls to perform the computation, but it demonstrates that it is possible to connect *popc* to the already existing infrastructure for OpenMP offloading in LLVM.

4 Conclusion and next steps

Although OpenMP intends to support all type of devices, the current specification has been too influenced by GPGPU-like systems. Therefore, most of the target regions are intended to support Single Program Multiple Data (SPMD) execution. Likewise, compiler implementations have exposed functions directly to the runtime API that follow this model. However, little support exists for devices that requires description of graphs. Although, the OpenMP specification does not restrict the use of `task` within `target`, description of static task creation, requires additional exploration and descriptions. Likewise, inter-device data distribution, or mapping clauses that do not map the whole array, but that can do some sort of data distribution is an area of improvement as well, if OpenMP wants to support these programming models. Finally, task placement (affinity) is not straightforward.

4.1 Limitations of this work

Similar to any other device currently supported by OpenMP offloading, there are features that this work does not currently support. Some of these are: virtual graphs, mapping of tensors and vertex to specific tiles, optimizations, or task level inter IPU programming. We have limited the definition of tasks to represent compute sets, therefore, cross dependencies between compute sets are not necessarily respecting OpenMP semantics. Finally, control flow within parallel regions has been restricted to represent the full parallel nature of compute sets. Finally, there are a large set of poplar SDK functions that have not been explored yet (e.g. vector transpose).

Some other limitations are more hardware constraints. First, given the small IPU memory, if streaming from the CPU host is not enabled during data mapping, it is not possible to fit really large applications. Another limitation is that the IPU architecture does not support double precision floating point units.

4.2 Future work

It is necessary to find applications that properly map to this new programming model. We believe that applications such as signal processing, and filtering, or those with long pipelines that could exploit streaming are good candidates. There are still open questions w.r.t. how to map programs to this architecture. How can we describe the characteristics of the architecture in a way that can engage teams.

5 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

References

[1] S. F. Antao, A. Bataev, A. C. Jacob, G.-T. Bercea, A. E. Eichenberger, G. Rokos, M. Martineau, T. Jin, G. Ozen, Z. Sura, T. Chen, H. Sung, C. Bertolli, and K. O'Brien, "Offloading support for openmp in clang and llvm," in *2016 Third Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, 2016, pp. 1–11.

[2] J. Doerfert, M. Jasper, J. Huber, G. K. Abdelaal, Georgakoudis, T. Scogland, and K. Parasyris, "Breaking the vendor lock — performance portable programming through OpenMP as target independent runtime layer," 2022.

[3] J. Doerfert, A. Patel, J. Huber, S. Tian, J. M. M. Diaz, B. M. Chapman, and G. Georgakoudis, "Co-designing an openmp GPU runtime and optimizations for near-zero overhead execution," in *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*. IEEE, 2022, pp. 504–514. [Online]. Available: https://doi.org/10.1109/IPDPS53621.2022.00055

[4] Graphcore, "BOW IPU Processor," https://www.graphcore.ai/bow-processors, online; accessed 14 Oct 2022.

[5] ——, "IPU Programming Guide," https://docs.graphcore.ai/projects/ipu-programmers-guide/en/latest/index.html, online; accessed 14 Oct 2022.

[6] ——, "Poplar Tutorial 3: Writting vertex code," https://github.com/graphcore/tutorials/tree/sdk-release-3.0/tutorials/poplar/tut3_vertices, online; accessed 14 Oct 2022.

[7] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the graphcore IPU architecture via microbenchmarking," *CoRR*, vol. abs/1912.03413, 2019. [Online]. Available: http://arxiv.org/abs/1912.03413

[8] Karl Freund and Patrick Moorhead, "THE GRAPHCORE SECOND GENERATION IPU," https://www.graphcore.ai/mk2-ipu-m2000-white-paper, online; accessed 14 Oct 2022.

[9] S. Tian, J. Huber, J. R. Tramm, B. M. Chapman, and J. Doerfert, "Just-in-time compilation and link-time optimization for openmp target offloading," in *OpenMP in a Modern World: From Multi-device Support to Meta Programming - 18th International Workshop on OpenMP, IWOMP 2022, Chattanooga, TN, USA, September 27-30, 2022, Proceedings*, ser. Lecture Notes in Computer Science, M. Klemm, B. R. de Supinski, J. Klinkenberg, and B. Neth, Eds., vol. 13527. Springer, 2022, pp. 145–158. [Online]. Available: https://doi.org/10.1007/978-3-031-15922-0_10

[10] H. Yviquel, L. Cruz, and G. Araujo, "Cluster programming using the openmp accelerator model," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, aug 2018. [Online]. Available: https://doi.org/10.1145/3226112

Deep Learning Driven Reaction Condition Predictions of Covalent Organic Frameworks

Ganesh Sivaraman, Lauren Valentino, Arvind Ramanathan, and Ian Foster

October 14, 2022

1 Introduction to the Science problem

Describe the science problem. What benefits will one have by leveraging the AI system for the science?

Covalent organic frameworks (COFs), an emerging class of materials with tunable physical and chemical properties that are ideal for gas adsorption/storage, filtration, separation, and catalysis. The structural versatility of COFs results in highly controllable structure and properties, providing numerous possibilities to create engineered pore environments (i.e., pore size, pore geometry, and chemical functionality). However, identifying COFs that both provide desired behaviors and are efficiently synthesizable is challenging. Quantum chemistry (QC) based computational techniques can be used to estimate properties such as formation and reaction energies, but are too expensive to permit exhaustive screening of large design spaces. Identification of synthesis routes is also complicated, as multiple parameters affect COF formation, including solvent, reactant concentration and ratio, temperature, and catalyst type and concentration, and the high temperature and long reaction times (>3 days) associated with typical synthesis routes make empirical (trial-and-error) screening of a wide range of combinations cost prohibitive. We propose instead to enable the AI-informed co-design of COFs synthesis pathways through AI driven synthesis planning and reaction condition optimization. We can bypass expensive QC calculation by means of AI/ML by leveraging the PyTorch framework.

2 Description of the AI model and implementation

Dataset To build the AI workflow we will use adopt the high throughput experiment dataset created by Perera et al. [1] for the Suzuki–Miyaura reactions. In the Figure. 1 variations of the ligands (11 in total), boronic acid (6) , aryl halide (4), bases (7), and solvents (4) lead to their database. We have adopted the version of this database curated by Saebi et al. [2] consists of a total of 4,543 measured yields.

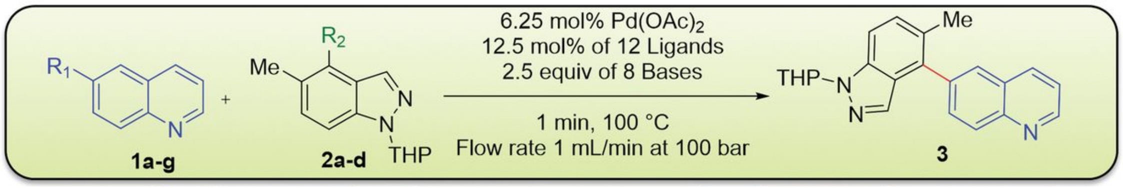


Figure 1: Suzuki-Miyaura cross-coupling reaction model

Machine learning reaction yield

Critical to achieving the goal of this LDRD is to develop a surrogate model that can bypass expensive quantum chemical calculation by directly map from input space of chemical reaction along with all experimental variables in the output space of reaction yields. For this task we adopt the Yield Graph Neural Network architecture (YieldGNN), based on Saebi et al. [2]. An overview of the model named YieldGNN is shown in Figure 2. YieldGNN learns reaction yield labels by combining the power of a GNN in capturing the higher-order neighbouring interactions in molecules involved in chemical reactions augmented that

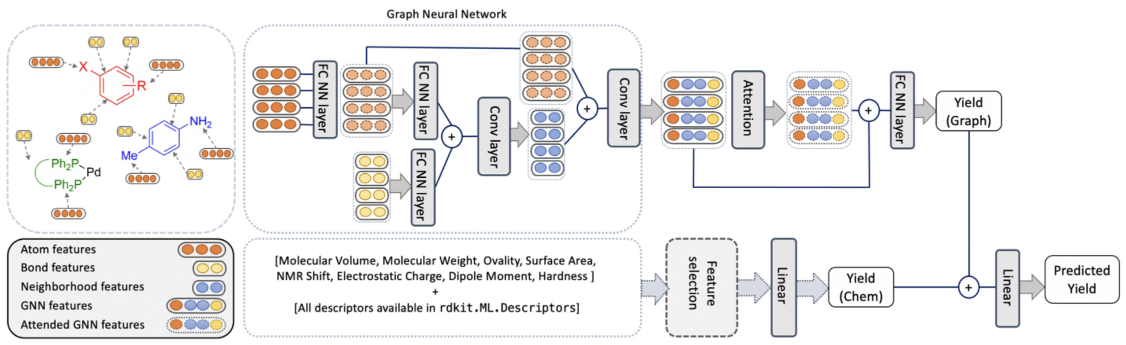


Figure 2: Overview of the YieldGNN model where the molecular features are learned by aggregating atom and bond features over the neighborhood (top part) and are combined with the RDKit chemical features (bottom part) to generate two yield scores Yield (Graph) and Yield (Chem). The two scores are passed through a linear layer to generate the final predicted yield .

with a complementary neural network that takes in to account physically/ chemically meaningful molecular properties. The GNN architecture is a highly expressive Weisfeiler-Lehman Networks [3, 4].

A list of the most relevant chemical features utilized are listed here 1) *Molecular features:* molecular volume, surface area, ovality, molecular weight, HOMO/LUMO Energy, electronegativity, hardness, and dipole moment. 2) *Atomic features:* Electrostatic charge and NMR shift. 3) *Reaction features:* Temperature, Reaction scale and volume for some of reactions

3 What was needed to get the model running on the AI Accelerator

We considered YieldGNN due to the practical consideration of being a fully PyTorch implementation [5]. We ported the code base for NVIDIA A100 as the initial starting point with the ultimate goal of porting the models to Graphcore. We will start working on the porting as soon as the Graphcore systems will be installed onsite at ANL and made accessible to the users. We are coordination with the ALCF datascience team for this step.

4 Performance Evaluation

YieldGNN model is ported on 1x NVIDIA A100 at the Swing LCRC cluster. A 70:30 split was used between training and validation dataset respectively. This resulted in assigning 3,180 reactions to model training and 1363 reaction to validation. The evolution of RMSE with respect to training epochs can be seen in Figures 3. On NVIDIA A100, the model trained for an average of 117 s/epoch. The model achieved a very competetive final R^2 of 0.82.

5 Conclusion and next steps

Please describe next steps planned, lessons learned and benefits to your science.

To conclude, we ported the YieldGNN model to NVIDIA A100. The model trained on the well known Suzuki-Miyaura reaction dataset reached a very competitive performance on reaction yield prediction.

Next steps. Work will be undertaken in close coordination to run the application on the Graphcore system to be made available for user in the near future. The work developed during this LDRD will be prepared for a manuscript publication with additional results for a real world COF dataset derived from the Reaxys database. We will perform reaction condition optimization for the best performing model.

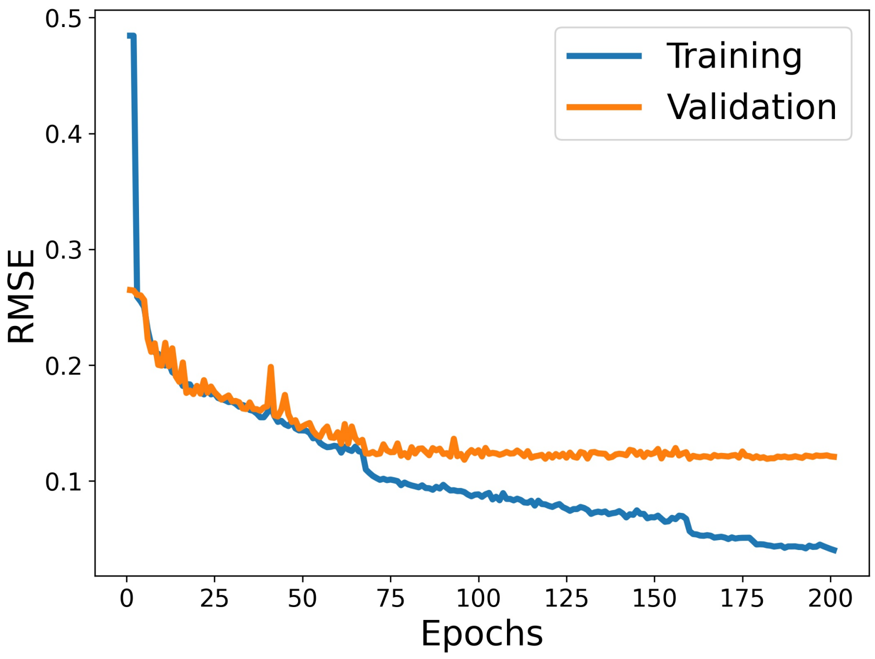


Figure 3: Evolution of the root mean sqaure error (RMSE) with respect to training epoch for YieldGNN run on NVIDIA A100. The validation errors are also shown for comparison purpose.

6 Acknowledgements

We gratefully acknowledge the the GraphCore AI testbed operated by the Argonne Leadership Computing Facility and the high-performance computing cluster Swing operated by the Laboratory Computing Resource Center at Argonne National Laboratory, both of which are DOE Office of Science User Facilities supported under Contract DE-AC02-06CH11357.We thank the ALCF datascience team for ongoing assistance. This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.

References

[1] Damith Perera et al. “A platform for automated nanomole-scale reaction screening and micromole-scale synthesis in flow”. In: *Science* 359.6374 (2018), pp. 429–434.

[2] Mandana Saebi et al. “On the Use of Real-World Datasets for Reaction Yield Prediction”. In: (2021).

[3] Tao Lei et al. “Deriving neural architectures from sequence and graph kernels”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2024–2033.

[4] Keyulu Xu et al. “How powerful are graph neural networks?” In: *arXiv preprint arXiv:1810.00826* (2018).

[5] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

Exploring Groq as a Real-time AI Inference Accelerator for Scientific Instruments

Kazutomo Yoshii (MCS), Marco Trovato (HEP), Anakha V Babu (XSD)

October 2022

1 Introduction to the Science problem

As the data rate from physics experiments continues to increase, sending all data downstream for analysis is becoming harder or even impossible. It is crucial to process data or make decisions close to the data source. We propose to characterize Groq as an AI accelerator candidate near scientific instruments, particularly considering data movement and I/O connectivity (e.g., host-PCIe accelerator).

Real-time AI technology that can be placed at the instrument edges is one of the important pieces for future autonomous laboratory environments, particularly high-energy physics (HEP) and X-ray science. For example, the frame rate of X-ray detectors will approach 1MHz soon, which will require micro-second-level inferences. Another example is that the CERN large hadron collider produces lots of data every 25 ns ($\mathcal{O}(MB)$), thus requiring prohibitive bandwidths ($\mathcal{O}(Pb/s)$). To keep bandwidths under control without impacting the experiment accuracy, ASICs and FPGAs are currently popular platform choices in HEP to pre-process the data; thanks to that technology, data paths can also be customized. In our previous study on Groq, we observed 18 usec/inference (computation only) / 50 usec/inference (with runtime overhead) for BraggNN batch inference with a Groq executable created from the PyTorch model via ONNX. In this proposal, we plan to exploit further optimization techniques, including native Groq APIs, targeting a couple of relevant AI models for HEP and X-ray science, and characterize the real-time inference capability of Groq. We also plan to discuss possible direct connectivity options from scientific instruments and RDMA-like operations with Groq. The outcome of this proposal is the performance characterization of Groq and the source codes of the optimized implementation of our selected scientific edge workloads. Additionally, through this opportunity, we will create a multidisciplinary collaboration among APS, HEP, and MCS divisions.

2 Description of the AI model and implementation

In this project, we focus on principal component analysis and the Sobel filter, important computation kernels needed in the real-time pipeline at the edge of scientific instruments.

2.1 Principal component analysis and lossy compression

Data volume increases in scientific instruments are becoming a critical issue and a huge bottleneck between the detector and downstream systems. One approach is to compress the data in the detector. Argonne has been working on an X-ray detector ASIC design with hardware lossless compressor [1, 2] that can keep up with the framerate and offers a moderate compression ratio (e.g., up to 10x). In some cases, we may need higher compression at the cost of quality. We have been looking at principal component analysis (PCA), an unsupervised machine learning technique, to reduce the data size. The forward path of PCA is relatively simple. The challenge for near-detector real-time applications is the performance. For example, each forward path requires 98M arithmetic operations for a 128x128 frame, which needs to be done within frames. In this report, we develop a PCA kernel on Groq, measure its performance, and estimate the maximum possible framerate with the current Groq TSP.

2.2 Sobel Filter

When an ionizing particle passes through a silicon sensor in HEP detectors, the charge is often deposited in more than one pixel. This is a direct consequence of the silicon sensor and particle characteristics. Moreover, in dense environments, where the average two-particle separation is comparable with the detector's granularity, overlapping clusters may occur, thus degrading the reconstruction performance. AI can be used to reconstruct clusters and correctly assign clusters to particles. Equipping the detector front-ends with the ability to form clusters of charges would allow front-ends to discard clusters from low-transverse momentum particles, which are expected to be composed of many pixels with charge above threshold. By doing so, only relevant information would be sent upstream, thus significantly reducing the output bandwidth. Such reduction will help mitigate the detector costs by having less demanding requirements on technology, and the data acquisition costs, by avoiding complex data encoding schemes, which require a lot of CPU power to be decoded. In this document we implement the Sobel Filter on Groq, as a first step towards HEP smart detectors.

3 What was needed to get the model running on the AI Accelerator

Since the original code for PCA-based lossy compression uses double precision and the matrix-multiply systolic accelerator in Groq TSP supports 8-bit integer and 16-bit floating point, we first need to evaluate the accuracy of the computation with different precision in software. We confirmed that PCA with 32-bit @ 32-bit matrix multiply had effectively no impact on the accuracy; however, 32-bit @ 16-bit matrix multiply yielded completely wrong results. This is primarily because the value range of the second matrix (PCA's encoding matrix) is way below the lower value of FP16. Fortunately, we found that a linear quantization on the encoding matrix allowed us to use FP16 for PCA.

4 Performance Evaluation

This section presents the performance comparison between Groq APIs and the performance analysis of PCA-based lossy compression and the Sobel filter.

4.1 Performance comparison between different APIs

Groq SDK offers different programming APIs to define a model. One way is to define a model using machine learning frameworks (PyTorch and Keras). This way, users need to 1) convert the model into a Groq program file via ONNX, a manual process, or 2) automatically convert the model into a program file using a newly released **GroqFlow**. Another way is to define a model using **Groq API**, which provides a basic set of neural network primitives, lower-level memory, and stream control primitives. For GroqFlow, the runtime is a simple method call with two options: 1) run with a single data and 2) run with a list of data (**run_abunch**). For Groq API, a simple way to run the model is to use **tsp_runner**, which involves the kernel in a blocking manner with suboptimal performance. To maximize the performance of the runtime, we need a **nonblocking** invocation, which requires us to write a decent amount of boilerplate codes.

Figure 1 includes the compute-only performance and the runtime performance that includes the overhead of invocation and data movement between the host and the accelerator on three different runtime methods: GroqFlow(run_abunch), Groq API (tsp_runner), Groq API (nonblocking). The main purpose of this comparison is to understand the runtime overhead of different runtime methods and the compute-only performance between GroqFlow and GroqAPI. The kernel we used for this benchmark is a 16-bit 128x128 matrix multiply, which directly fits into Groq's systolic matrix multiply unit. The compute-only performance is obtained from the *iop-utils* command. Even with this simple workload, we observe that Groq API yields 2.3x better performance than Groq Flow (the blue bars in Figure 1). In either case, the compute-only performance of Groq is impressive: Groq API reaches 4.8 TFlops for FP16. However, Groq TSP is a PCIe-attached accelerator and we cannot ignore the overhead from PCIe, data movement, and software runtime code. The green bars in Figure 1 show the average runtime performance of each forward path on three different API/Runtime combinations. As you can see, there is a significant runtime overhead, even with the most optimized runtime. At this point, we have no way to measure the breakdown.

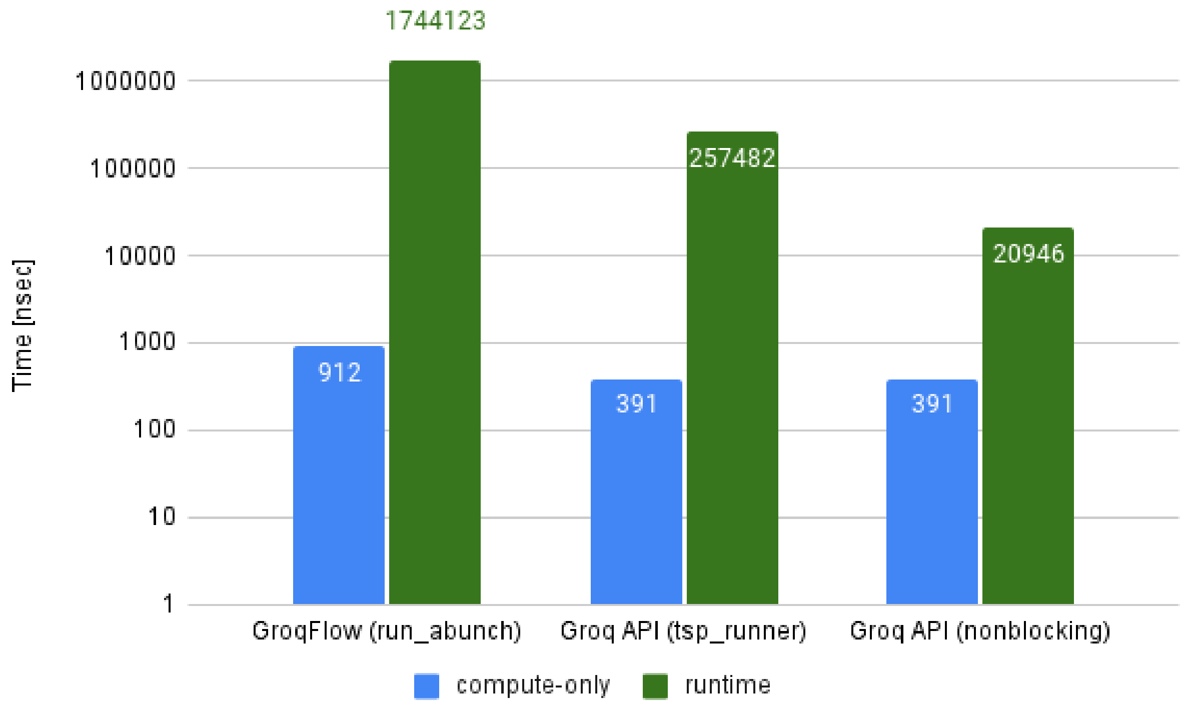


Figure 1: Runtime and API performance comparison

4.2 PCA-based lossy compression

We have implemented the forward path of PCA-based lossy compression using Groq API with the non-blocking runtime. The target input data size is single channel 128x128 images. The data type we chose is 16 bits (we confirmed that 16-bit datatype is acceptable for PCA with a linear quantization applied). The variable parameter in this experiment is the size of the encoding matrix, which is essentially weight size. The larger the encoding matrix higher the compression quality but, the lower the compression ratio. We wanted the encoding matrix size up to 3,000; however, the constant tensor allocation for this size failed. We found that the automatic memory allocation works up to 1,200. Figure 2 includes both the compute-only and runtime performance. Every invocation transfers 32 KB of data (16-bit 128x128 image); approx. 13 GB/s of bandwidth on PCIe. Groq-accelerated PCA lossy compression reaches 167 KFPS with 1 and 52 KFPS with 1,200 (2.3 TFlops). These are excellent performance numbers; unfortunately, it is still far from the 1 MPFS target.

4.3 Sobel filter

Sobel Filter is expected to run over large pixel matrices (800×800 pixels) deployed in the upgraded version of the ATLAS detector at CERN [3]. A $N \times N$-pixel-sized kernel is used, where N is an integer of choice. A simplified Sobel Filter was implemented by using the Groq API Neural Network library and successfully compared to the equivalent NumPY solution: at this stage the implementation consists of a 2D convolution of the $N \times N$-pixel-sized portion of the pixel matrix with the kernel. For this study each pixel contains an information that can be encoded in 16 bits. For 3×3-size kernels the convolution is iterated

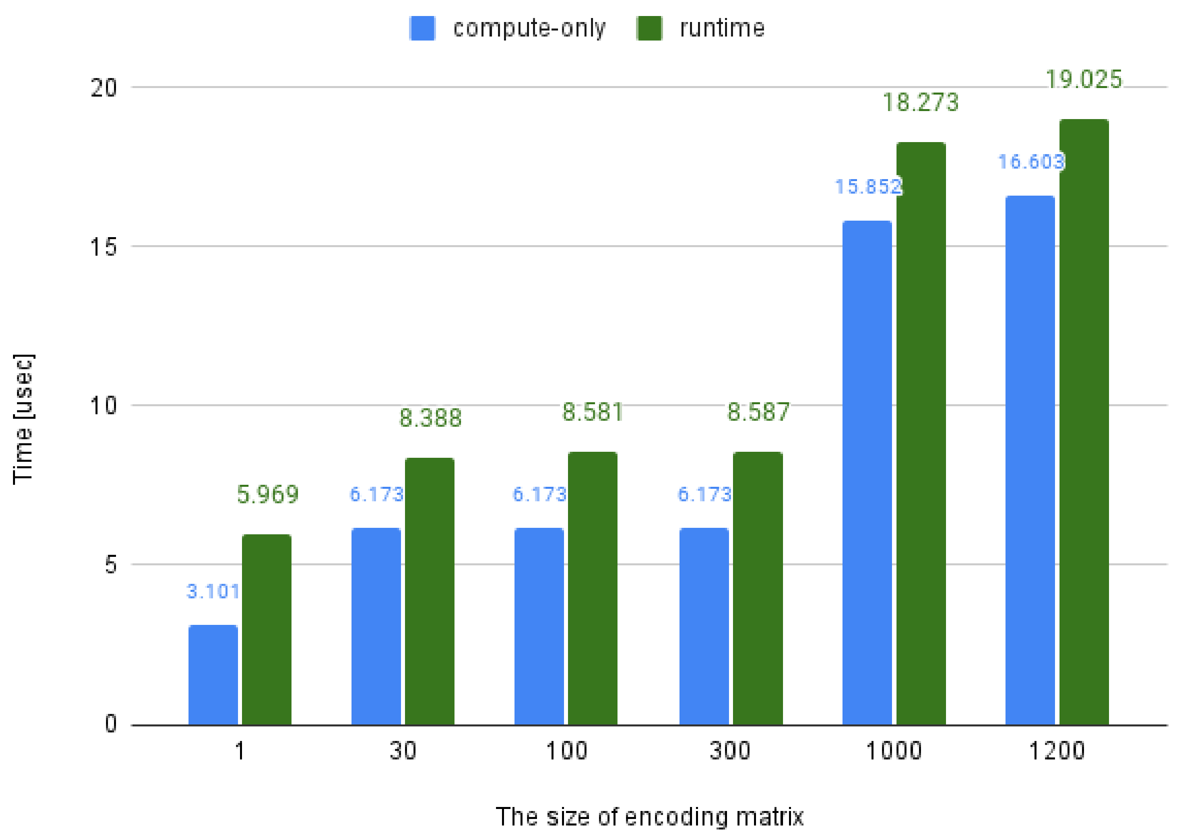


Figure 2: PCA-based lossy compression for 128x128 images

800×800 = 640, 000 times . A single convolution product takes 91 cycles. The overall computational time taken by the Groq chip is 91×640,000×1/900 MHz ∼ 64.7 ms, where a clock frequency of 900 MHz has been assumed. This time does not include transferring of data from and to the chip.
In order to characterize and better understand the chip we scan the kernel size and change data types. From the results shown in 3 we infer that the number of cycles increases linearly (slope of 1) for 8-bit integer data types, after an initial setup cost from loading the weight matrix. For 16-bit floating data types, the same linear relationship holds for matrix sizes up to $N = 8$. An increase of 30 cycles happens at $N = 9$. Starting from $N = 20$ the linear relationship is with a steeper slope of 2. It should be noticed that the 16-bit floating point data is more efficient in terms of throughput.
The Sobel filter implementation still misses the final step of summing each element of the convoluted matrix. That is work in progress

5 Conclusion and next steps

As some of us are new to Groq, we spent some time researching the documentation that would allow understand the Groq architecture. Thanks to the documentation and the 2021 tutorial, we could start implementing our models. While implementing the models, we learned the chip's computation power and flexibility, and we plan to improve our skills on the chip in the future if time allows. For what concerns the Sobel Filter, we would like to finalize the design and directly compare the Groq performance with the

[0]3×3-size kernel is an arbitrary choice, which was made while waiting for simulation results to point to an optimal size. Pixel matrix has been padded with zeros, thus resulting in 802 × 802-size matrix

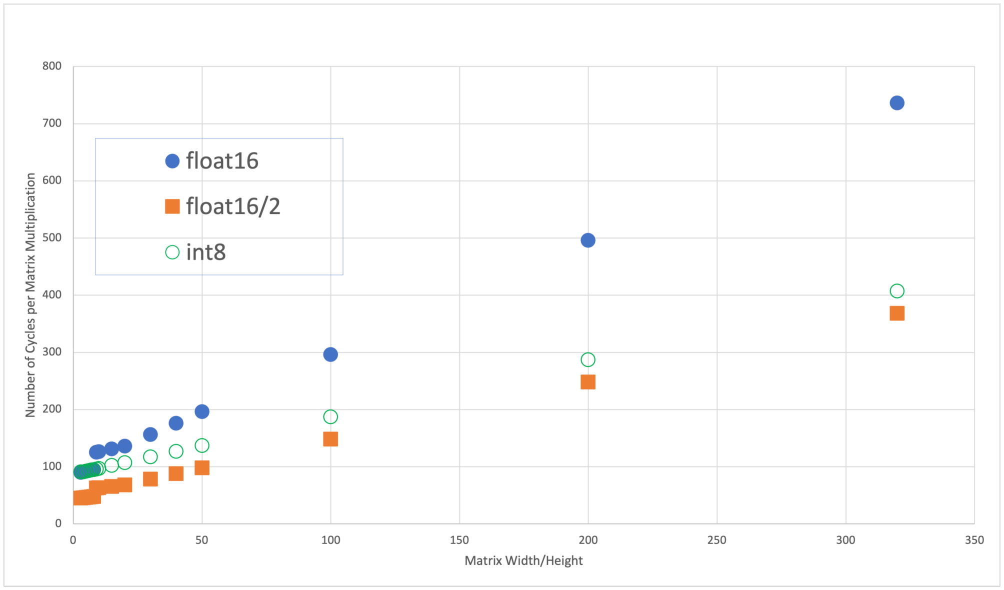


Figure 3: Number of cycles per matrix multiplication as function of matrix width or height. *Float*16 (Blue) and *int*8 (Gray) data types are also compared. Number of cycles represented with orange dots, which are exactly half of the cycles when using *float*16 data types, are shown to have a fairer comparison with Int8 from a throughput standpoint.

CPU and FPGA counterparts that we have/will have at hand. We believe there is still room for improving the performance of our target workload, as we don't have a complete understanding of Groq API yet, in particular, memory allocation and streaming control. We are interested in optimizing our implementation by applying advanced Groq API techniques and testing the C-based runtime API that Groq will release soon, which is expected to reduce the runtime overhead. Additionally, we continue to search AI architecture candidates that can be embedded directly in scientific detector chips (possibly as a chiplet).

6 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. We thank the X-ray detector group led by Antonino Miceli for fruitful discussions on X-ray detector systems and their future direction. We thank Groq for the numerous advances and suggestions on their AI accelerators.

References

[1] Mike Hammer, Kazutomo Yoshii, and Antonino Miceli. Strategies for on-chip digital data compression for x-ray pixel detectors. *arXiv preprint arXiv:2006.02639*, 2020.

[2] Kazutomo Yoshii, Rajesh Sankaran, Sebastian Strempfer, Maksim Levental, Mike Hammer, and Antonino Miceli. A hardware co-design workflow for scientific instruments at the edge. In *Smoky Mountains Computational Sciences and Engineering Conference*, pages 209–226. Springer, 2021.

[3] Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System. Technical report, CERN, Geneva, 2017.

Technical Report for LDRD Expedition Project
Deep Learning Accelerated X-ray Data Analysis for Experiment Steering
BraggNN inference evaluation using Groq

Zhengchun Liu, Rajkumar Kettimuthu
Data Science and Learning division,
Argonne National Laboratory

October 2022

1 Introduction to the Science problem

Experimental protocols, e.g., high energy X-ray diffraction microscopy (HEDM) and iterative Computed Tomography (CT), at synchrotron light sources typically process and validate data only after an experiment has completed, which can lead to undetected errors, lost resolution for deformation events and cannot enable online steering. Real-time data analysis can enable both detection of, and recovery from, errors, and optimization of data acquisition. However, modern scientific instruments, such as detectors at synchrotron light sources, can generate data at GBs/sec rates. Conventional data processing methods usually require considerable computational resources which are usually neither available near the data acquisition machine, nor ready at remote supercomputer centers in a on-demand fashion. Machine Learning based surrogate usually can run much faster on GPUs but there is a tradeoff between throughput and latency.

In the case of data analysis for experiment steering, as latency is the primary consideration, we need to run model inference with very small batch size (accumulating large batch will increase latency linearly). Thus, one needs to over-provision GPU resource significantly so as to meet the requirement of both latency and throughput. As Groq's deterministic single-core streaming architecture is designed for both low latency and throughput for real-time AI, we proposed to explore the fitness of Groq, as an inference engine, to enable low latency and high throughput inference for experiment steering.

In this report, we share our evaluation on the inference performance of Groq for BraggNN, a deep learning based solution to analyze Bragg diffraction data faster for HEDM. We picked BraggNN because HEDM experiments generates 100s of diffraction frames at APS now (1000s with APS-U), each frame contains dozens to thousands of diffraction spots, there is a practical need to process these big data in realtime to mitigate storage challenge and enable experiment steering. Thus, although BraggNN is not a widely used benchmark application, insights for BraggNN benchmarks on the Groq system can practically benefit diffraction image techniques at APS.

2 Description of the AI model and Benchmark data

2.1 BraggNN

X-ray diffraction based microscopy techniques such as HEDM rely on knowledge of the position of diffraction peaks with high precision. These positions are typically computed by fitting the observed intensities in area detector data to a theoretical peak shape such as pseudo-Voigt. As experiments become more complex and detector technologies evolve, the computational cost of such peak detection and shape fitting becomes the biggest hurdle to the rapid analysis required for real-time feedback during in-situ experiments.

BraggNN [1], as shown in Figure 1, is a deep learning model designed to localize Bragg peak positions much more rapidly than conventional pseudo-Voigt peak fitting. Recent advances in deep learning method implementations and special-purpose model inference accelerators allow BraggNN to deliver enormous performance improvements relative to the conventional method, running, for example, more than 200 times faster than a conventional method on a consumer-class GPU card with out-of-the-box software.

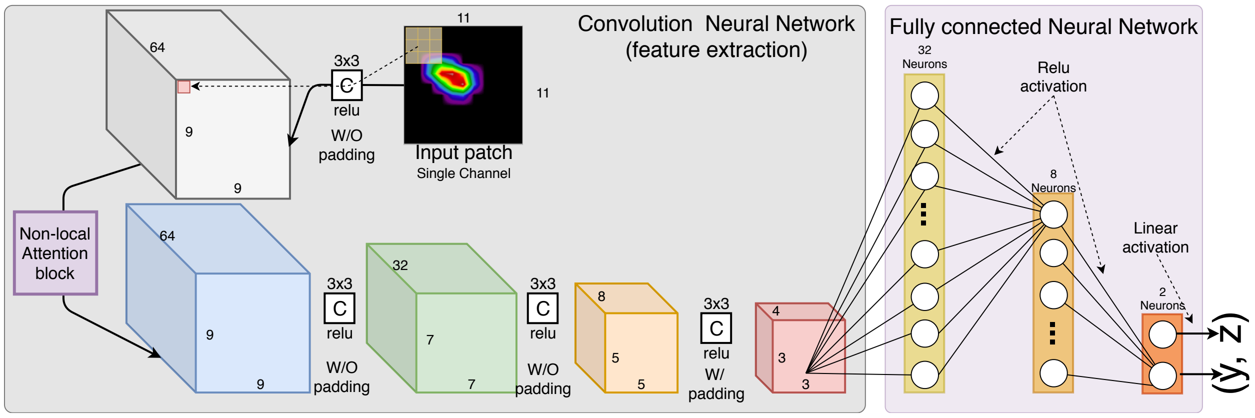


Figure 1: Architecture of the BraggNN model.

Our recent work shows that the BraggNN inference on an edge computing device, Jetson AGX, can process diffraction data at a rate of 10 frames per second. As a comparison, one minutes of data will need more than an hour using conventional methods on a server with 80 CPU cores.

2.2 Dataset

We used 13,792 samples from a real HEDM experiment, each input is 11x11 pixels as shown in Figure 2, as the benchmark dataset for this study. Our evaluation includes both model accuracy (i.e., accuracy loss from inference at lower precision must be practically acceptable before doing any throughput evaluation) and throughput under different batch size. Euclidean distance between BraggNN prediction and ground truth is

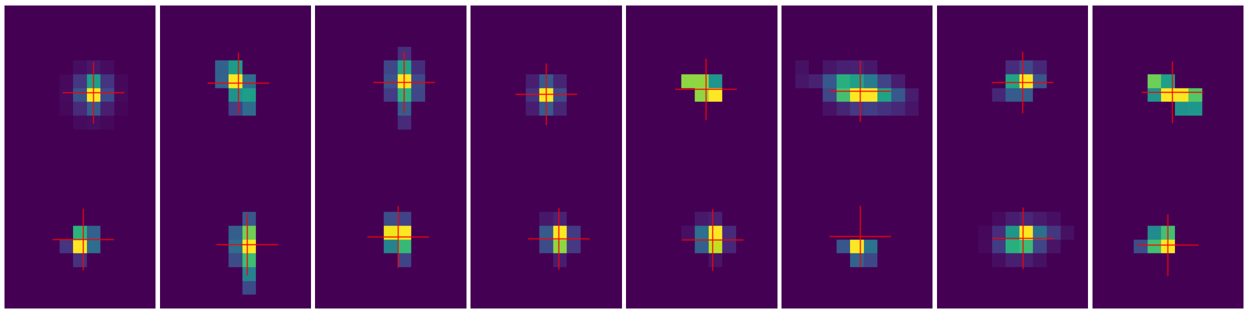


Figure 2: Illustration of inputs (image patch) and targets (location of the red cross).

the metric for error evaluation. Distribution (histogram) of the error is used for overall error evaluation.

3 What was needed to get the model running on the AI Accelerator?

It is straightforward to run BraggNN on Groq for inference. Steps to get BraggNN running on Groq includine:

1. Export trained model to ONNX, as BraggNN was implemented using Pytorch, we use `torch.onnx.export` to export Torch model into ONNX. One important thing we need
2. Cast FP32 weights into FP16 using `convert_float_to_float16` from `onnxmltools` package.

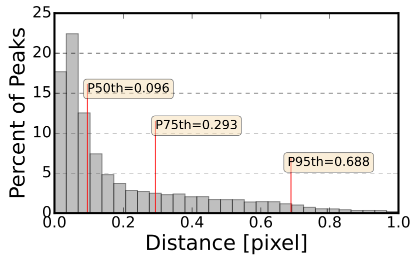
3. As the accuracy loss when use Int8 is not acceptable, we compile the FP16 directly without quantization. `groq-compiler` is used to compile ONNX model into Alan Assembly program and then use `aa-latest` to compile the Alan Assembly program to a Groq executable.

4. Building a Grop runner using `groq.runner.create_tsp_runner` from the compiled Groq executable and run inference using the runner.

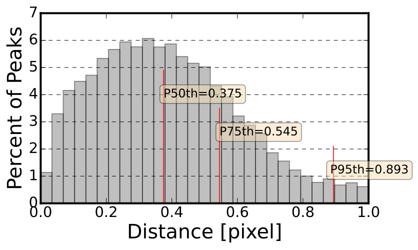
4 Performance Evaluation

4.1 Accuracy

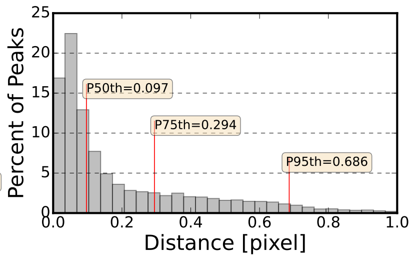
Figure 3 compares the prediction error distribution using different platforms (CPU and Groq) and computing precision (Int8 vs. FP32 vs. FP16). The latency, per sample on average is 127 μs, 11.7 μs and 8.4 μs for CPU-FP32, Groq-I8 and Groq-FP16 individually. Running at Int8 with quantization is surprisingly slower than directly at FP16. This is possibly because of the extra cost on quantization and dequantization operators.



(a) CPU with FP32



(b) Groq with Int8



(c) Groq with FP16

Figure 3: Distribution of difference between peak position located by BraggNN and conventional pseudo-Voigt fitting, using different platform and precision. P_{nth} denotes the n_{th} percentile.

As one can see from Figure 3 that, the accuracy loss from running at Int8 with quantization is too big to be acceptable. The difference between FP16 and FP32 is less than 1% in average, which can be considered as acceptable from our perspective.

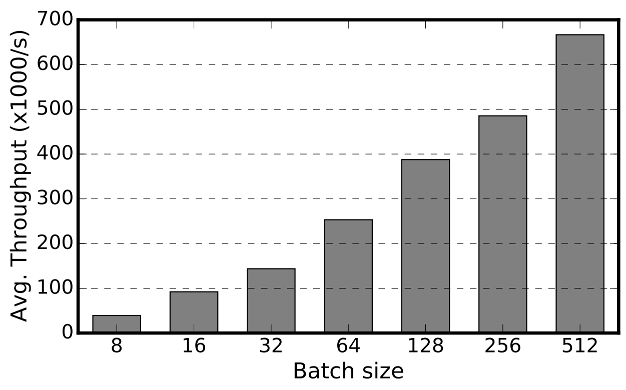
4.2 Latency/Throughput

Figure 4b presents the inference throughput as a function of batch size. The throughput does increase with batch size. However, due to the limitation of on-chip SRAM size, the maximum batch size Groq can run for BraggNN is 64 (only numbers of the order of 2 are explored). Till batch size of 64, we did not see saturation on throughput.

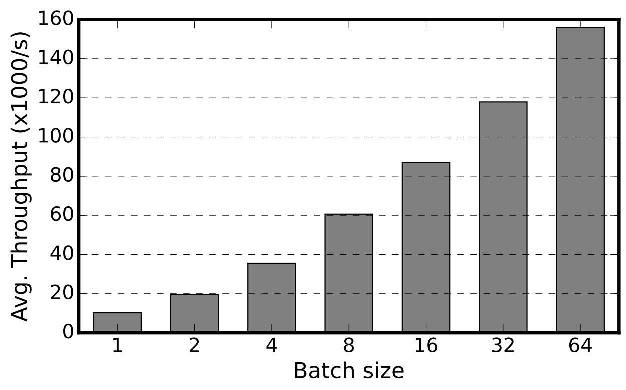
From another study, we also benchmarked the sample model on NVIDIA V100 via TensorRT with FP32, shown in Figure 4b. The common mini batch size, between these two benchmarks are 8, 16, 32 and 64. As a (unfair) comparison, V100 is slower than Groq when batch size is small. V100 throughput increase much faster with batch size compare with Groq. For example, Groq throughput is 1.54× of V100 when batch size is 8, while V100 become 1.62× of Groq when batch size increases to 64. We do note that, this is not a systematic benchmark for comparison purpose.

5 Conclusion and next steps

Our practice and experiment shows that, Groq system is relatively easy to use, it does not require that much effort to make our model work on it. Post-quantization with Int8 cannot deliver acceptable accuracy, we thus have to use FP16 without model quantization. Quantization aware training may help but we do not have enough time to practice. Moreover, we also noticed that quantization is not always faster for BraggNN because of the excessive quantization and de-quantization operations. The comparison with NVIDIA V100



(a) NVIDIA V100 with FP32



(b) Groq with FP16

Figure 4: Inference throughput as a function of batch size.

shows that Groq seems do better on small batch size as claimed by Groq system Inc although our comparison is not holistic.

As for future work, we are currently planning to work with Groq team to further optimize the inference performance, and we also plan to integrate one Groq chip directly with the CookieBox detector at SLAC (possibly at APS).

6 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This material is based upon work supported by a Laboratory Directed Research and Development (LDRD) Program at Argonne National Laboratory.

References

[1] 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*, 9(1), 2022.

Appendix

We append the complete source code we used for this study. The dataset and pre-trained model are available upon request.

```
from model import model_init, BraggNN
import torch, argparse, os, time, sys, logging, h5py, time
from torch.utils.data import DataLoader, Dataset
import numpy as np
from subprocess import check_output

class BraggNNDataset(Dataset):
    def __init__(self, ifn):
        with h5py.File(ifn, 'r') as h5:
            self.patches = h5['patch'][:]
            self.peakLoc = h5['ploc'][:] - 0.5
            self.psz = self.patches.shape[-1]

    def __getitem__(self, idx):
        return self.patches[idx], self.peakLoc[idx]

    def __len__(self):
        return self.patches.shape[0]

def s2ituple(s):
    return tuple(int(_s) for _s in s.split('_'))

def pth2onnx(args):
    model = BraggNN(imgsz=args.psz, fcsz=args.fcsz)
    model.load_state_dict(torch.load(args.mdl, map_location=torch.device('cpu')))
    for param in model.parameters():
        param.requires_grad = False

    dummy_input = torch.randn(args.mbsz, 1, args.psz, args.psz, dtype=torch.float32, device='cpu')

    model_script = torch.jit.trace(model.cpu(), dummy_input)
    script_fname = args.mdl.replace('.pth', '_script.pth')
    torch.jit.save(model_script, script_fname)

    input_names  = ('patch', )
    output_names = ('ploc',  )

    onnx_fn = args.mdl.replace(".pth", ".onnx")
    torch.onnx.export(model, dummy_input, onnx_fn, verbose=False, opset_version=13,\
                      input_names=input_names, output_names=output_names)
    return onnx_fn, script_fname

def onnx_cast_to_fp16(fp32_onnx):
    from onnxmltools.utils.float16_converter import convert_float_to_float16
    from onnxmltools.utils import save_model
    from onnxmltools.utils import load_model

    fp32_model = load_model(fp32_onnx) # Load model to convert
    fp16_model = convert_float_to_float16(fp32_model) # Convert fp32 model to fp16

    fp16_onnx = fp32_onnx.replace('.onnx', '_fp16.onnx')
    save_model(fp16_model, fp16_onnx) # Save fp 16 model
    return fp16_onnx

def quantize(args, inp_onnx):
    import onnxruntime
    from onnx import helper, save, shape_inference, load
    from onnxruntime.quantization import quantize_static, CalibrationDataReader, QuantType

    class DataReader(CalibrationDataReader):
        def __init__(self, model_fname):
            name = onnxruntime.InferenceSession(inp_onnx, None).get_inputs()[0].name

            with h5py.File(args.ih5, 'r') as h5:
                patches = h5['patch'][:]
            # probably no need to use all esp when data is big
            nb = patches.shape[0] // args.mbsz
            self.data = ( {name: patches[i*args.mbsz:(i+1)*args.mbsz]} for i in range(nb) )

        def get_next(self):
            return next(self.data, None)
    oup_onnx = inp_onnx.replace('.onnx', '_quantized.onnx')
    quantize_static(model_input=inp_onnx,
                    model_output=oup_onnx,
                    calibration_data_reader=DataReader(inp_onnx),
                    activation_type=QuantType.QUInt8,
                    weight_type=QuantType.QInt8,
                    op_types_to_quantize=["Conv", "Relu", "MatMul", "Linear"])

    save(shape_inference.infer_shapes(load(oup_onnx)), oup_onnx)
    return oup_onnx

def compile(onnx):
    iop_path = onnx.replace('.onnx', '.iop')
    aa_path  = onnx.replace('.onnx', '.aa')

    # Compile quantized ONNX model to Alan Assembly
    compile_output = check_output(["groq-compiler", "-All", onnx])
    logging.info(compile_output.decode("utf-8"))

    # Compile Alan Assembly to TSP machine code
    assembly_output = check_output(["aa-latest", "-i", aa_path, "--output-iop", iop_path, "--no-metrics"])
    logging.info(assembly_output.decode("utf-8"))

    return iop_path

def cpu_inference(args, script_fname):
    ds = BraggNNDataset(args.ih5)
    test_dl = DataLoader(dataset=ds, batch_size=args.mbsz, shuffle=False, drop_last=True, pin_memory=True)
    psz = ds.psz

    model = torch.jit.load(script_fname, map_location='cpu')
    for param in model.parameters():
        param.requires_grad = False

    pred, gt = [], []
    inference_tick = time.time()
    for X_mb, y_mb in test_dl:
        with torch.no_grad():
            pred_val = model.forward(X_mb).cpu().numpy()
        pred.append(pred_val)
        gt.append(y_mb.numpy())
    time_on_inference = time.time() - inference_tick

    pred = np.concatenate(pred, axis=0)
    gt   = np.concatenate(gt,   axis=0)
    l2norm_val = np.sqrt(np.sum((gt - pred)**2, axis=1)) * ds.psz
    logging.info('[Valid] l2-norm of %5d samples: Avg.: %.4f, 50th: %.3f, 75th: %.3f, 95th: %.3f, 99.5th: %.3f (pixels)
                    (l2norm_val.shape[0], l2norm_val.mean()) + tuple(np.percentile(l2norm_val, (50, 75, 95, 99.5))) ) )

    logging.info("Inference for %d samples (%s) using mbsz of %d on CPU, took %.3f seconds (%.2fus/sample)" % (\
                 gt.shape[0], args.ih5, args.mbsz, time_on_inference, 1e6*time_on_inference/gt.shape[0]))

    with h5py.File('infer-res-cpu.h5', 'w') as h5fd:
        h5fd.create_dataset('prediction' , data=pred*psz)
        h5fd.create_dataset('groundtruth', data=gt  *psz)

def groq_inference(args, iop_fname):
    from groq.runner import tsp

    with h5py.File(args.ih5, 'r') as h5:
        patches = h5['patch'][:].astype(np.float16)
        peakLoc = h5['ploc'][:] - 0.5
    psz = patches.shape[-1]
    nb = patches.shape[0] // args.mbsz
    # static mb size, must drop last
    patches = patches[:nb*args.mbsz]
    gt      = peakLoc[:nb*args.mbsz]

    runner = tsp.create_tsp_runner(iop_fname)

    # Run inference on chip
    pred = []
    inference_tick = time.time()
    for i in range(nb):
        x = patches[i*args.mbsz:(i+1)*args.mbsz].flatten()
        _pred = runner(patch=x)['ploc']
        pred.append(_pred)
    time_on_inference = time.time() - inference_tick

    pred = np.concatenate(pred, axis=0)

    l2norm_val = np.sqrt(np.sum((gt - pred)**2, axis=1)) * psz
    logging.info('[Valid] l2-norm of %5d samples: Avg.: %.4f, 50th: %.3f, 75th: %.3f, 95th: %.3f, 99.5th: %.3f (pixels)
                (l2norm_val.shape[0], l2norm_val.mean()) + tuple(np.percentile(l2norm_val, (50, 75, 95, 99.5))) ) )

    logging.info("Inference for %d samples (%s) using mbsz of %d on Groq, took %.3f seconds (%.2fus/sample)" % (\
             gt.shape[0], args.ih5, args.mbsz, time_on_inference, 1e6*time_on_inference/gt.shape[0]))

    with h5py.File('infer-res-groq.h5', 'w') as h5fd:
        h5fd.create_dataset('prediction' , data=pred*psz)
        h5fd.create_dataset('groundtruth', data=gt  *psz)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Bragg peak finding for HEDM.')
    parser.add_argument('-mbsz', type=int, default=32, help='mini batch size')
    parser.add_argument('-fcsz', type=s2ituple, default='16_8_4_2', help='size of dense layers')
    parser.add_argument('-ih5',  type=str, default='valid-ds-psz11.h5', help='input h5 file')
    parser.add_argument('-mdl',  type=str, default='0center-gpu.pth', help='model weights')
    parser.add_argument('-psz',  type=int, default=11, help='inp patch size')

    args, unparsed = parser.parse_known_args()

    if len(unparsed) > 0:
        print('Unrecognized argument(s): \n%s \nProgram exiting ... ... ' % '\n'.join(unparsed))
        exit(0)

    logging.basicConfig(filename='inference.log', level=logging.INFO)
    logging.getLogger().addHandler(logging.StreamHandler(sys.stdout))

    onnx_fp32, script_fname = pth2onnx(args)

    onnx_fp16 = onnx_cast_to_fp16(onnx_fp32)

    # onnx_i8   = quantize(args, inp_onnx=onnx_fp32)
    # iop_fname= compile(onnx=onnx_i8, )

    iop_fname= compile(onnx=onnx_fp16, )

    # iop_fname = '0center-gpu_quantized.iop'
    groq_inference(args, iop_fname)

    # cpu_inference(args, script_fname)
```

Machine learning at the edge for real-time analysis in X-ray ptychography enabled by hardware AI accelerators

Anakha V Babu, Tao Zhou, Saugat Kandel, Zhengchun Liu, Antonino Miceli, Mathew J. Cherukara

October 2022

1 Introduction to the Science problem

Coherent diffraction imaging (CDI) is a lensless technique that is widely used for nanoscale imaging of the sample (2D or 3D) from the measured diffraction pattern. X-ray ptychography is one of the CDI approaches that relies on the high overlap of the adjacent illuminated regions for creating a large field of view in the imaging process. This type of data acquisition can lead to extremely large experimental dataset sizes. The collected diffraction patterns are then used for reconstructing (phase retrieval) real-space 2D view of the sample. Traditionally, the reconstruction process depends on iterative techniques, however these techniques can have long turnaround time due to dataset sizes, which can limit the experimental capabilities such as real-time experimental steering and low-latency monitoring. The conventional iterative methods for phase reterieval are compute-intensive and cannot keep up with current ptychography data acquisition speed rates (32 Gbps for a 1M detector running at 1 kHz). Moreover, the data acquisition rates get exemplified especially after the APS upgrade, where the coherent flux is expected to increase by a factor of 100x.

Machine learning (ML) based alternate approaches have been shown to provide a 100x acceleration in phase retreival for ptychography. Here we adopted a slightly modified architecture of the neural network, PtychoNN discussed in [1] for real-time feed back at the edge. We have developed a deep learning accelerated workflow shown in Fig. 1, for X-ray ptychography coherent diffraction imaging which is fast enough to keep up with the data and provide accurate and real-time image reconstructions [2]. The automated workflow relies on training the neural network continually as the experiment progresses and the ability to perform faster training is impactful as the edge device will be updated with a better model sooner, providing accurate reconstructions at very early stages of the experimental scan. Here we used NVIDIA's Jetson platform at the edge for inference predictions on the live streamed diffraction patterns from the detector. In addition to streaming the data to the edge device over the network, the workflow concurrently saves the diffraction patterns for training at the end of the experimental scan. Though NVIDIA GPUs were used for training in [2], dedicated AI hardware accelerators can potentially offer faster training times and hence we explored Cerebras (CS-2) for this work.

2 Description of the AI model and implementation

Here we used a slighlty modified architecture of PtychoNN discussed in [1], which is a convolutional autoencoder-decoder network with 2D convolution operations, max pooling, ReLU activations, upsampling, and zeropadding. Originally, we used a PyTorch based implementation for the neural network with input/output sizes of 128×128. The network has a computational complexity of 320 MMacs and 730k trainable parameters. After APS-Upgrade, the computational complexity of the neural network for ptychography is expected to hit ∼100GMACs with ∼10 M trainable parameters.

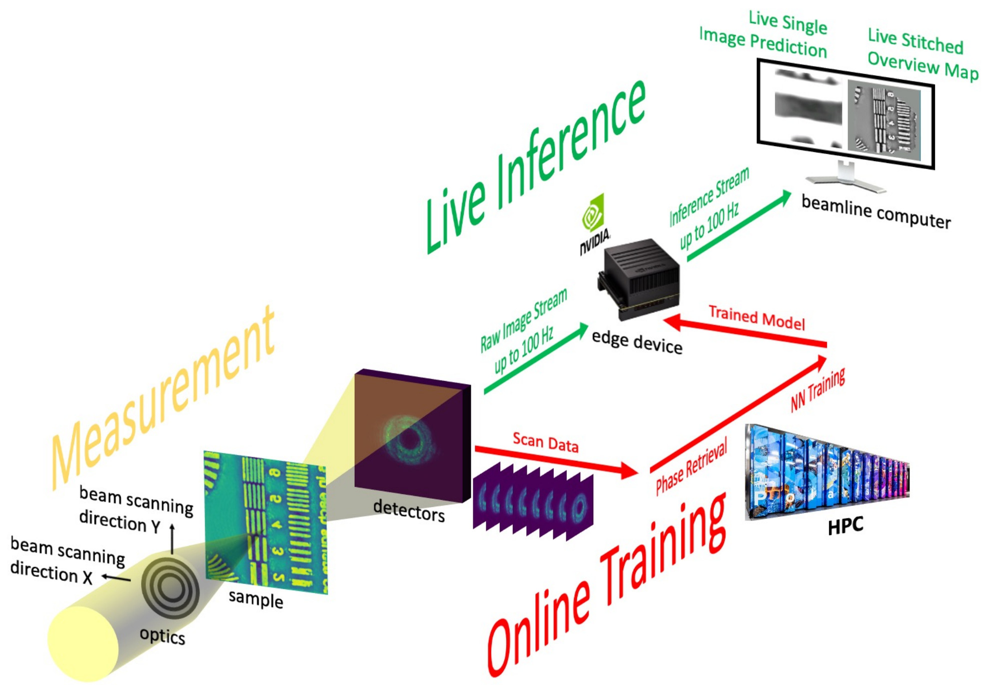

Live Single Image Prediction
Live Stitched Overview Map
Live Inference
Inference Stream up to 100 Hz
beamline computer
NVIDIA
Raw Image Stream up to 100 Hz
edge device
Trained Model
Measurement
NN Training
Scan Data
Phase Retrieval
beam scanning direction Y
detectors
HPC
beam scanning direction X
sample
optics
Online Training

Figure 1: Illustration of deep learning accelerated workflow for real-time streaming ptychography imaging [2].

3 What was needed to get the model running on the AI Accelerator

We adopted an approach of modifying the supported examples in the modelzoo shared by the CS-team instead of developing everything new. As the neural network in our study is a convolutional autoencoder network we used the TensorFlow implementation of the Autoencoder (AE) example from the anl shared repository. The first step is to generate the tfrecords for the training and test data using the script **writetfrecords.py** by making appropriate changes. Changes to the model was not necessary as the AE example supported all the operations in our neural network. The main challenge was to run the evaluation of the trained model on the test data. At the time of writing the report, there was no support for evaluation of the trained model for AE on the CS system. The CS team provided instructions for performing the evaluation on the CPU and we ran the evaluation on the test data successfully on the host. However, the CPU evaluation is very slow ($\sim$ 20 mins) and hence, it is necessary to have eval support on the wafer scale engine in the future.

4 Performance Evaluation

The modified PtychoNN is trained using the CS-2 system on a dataset having $\sim$ 100 experimentally acquired scans with a batch size of 256. The training is performed over 5000 steps using adam learning rate optimizer. The wall clock time for the training is observed to be $\sim$ **15** minutes. The maximum chunk of time is spent on compilation and CS-2 fabric programming with the actual neural network training time being **12.5** seconds. We anticipate the compilation and CS-2 programming will be observed only once at the beginning of the training and hence significant acceleration can be achieved in training times when the model is trained continually. We compared the training times of CS-2 against NVIDIA's A100 GPUs and is listed in Table 1. The number of GPUs is indicated in the curved parenthesis along with the training times. It can be noticed that CS-2 is three times faster compared to using 2 GPUs for training, and using 8 GPUs can eventually match up with the AI system.

Table 1: Approximate training times (wall clock time) using NVIDIA GPUs and CS-2

	# GPUs	CS-2
Training Times	45mins (2)	15mins
Training Times	15mins (8)	15mins

5 Conclusion and next steps

We successfully trained the neural network on the CS-2 system. We also hope to extend this work by proposing the following next steps if time permits.

1. **Conversion of trained TF checkpoint to ONNX**: To implement the inference on the edge device, we need to convert the TF trained checkpoint to ONNX format. This conversion is typically recommended on the same platform as training to avoid conflicting issues with TF versions and other underlying architecture differences. As of now, the CS-2 system doesn't support installing packages like tf2onnx on singularity shell and hence we tried to perform the conversion in the Jetson device (AGX Orin). However this conversion was not successful and CS team suggested to convert to TF savedmodel before attempting an ONNX conversion. Unfortunately, this step also did not succeed due to TF version issues. We hope to solve this problem given more time.

2. **Simulate the workflow shown in Fig. 1 using CS-2 for training**: Once step 1 is successfully achieved, the entire AI-accelerated workflow can be simulated using the simulated detector with ALCF resources for performing the conventional reconstructions, CS-2 for training and push the trained model to the edge device (Jetson Orin) over the network.

6 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] Mathew J. Cherukara, Tao Zhou, Youssef Nashed, Pablo Enfedaque, Alex Hexemer, Ross J. Harder, and Martin V. Holt. AI-enabled high-resolution scanning coherent diffraction imaging. *Applied Physics Letters*, 117(4):044103, 2020.

[2] Anakha V Babu, Tao Zhou, Saugat Kandel, Tekin Bicer, Zhengchun Liu, William Judge, Daniel J. Ching, Yi Jiang, Sinisa Veseli, Steven Henke, Ryan Chard, Yudong Yao, Ekaterina Sirazitdinova, Geetika Gupta, Martin V. Holt, Ian T. Foster, Antonino Miceli, and Mathew J. Cherukara. Deep learning at the edge enables real-time streaming ptychographic imaging, 2022.

Accelerating training and inference for machine learned force fields

Noah Paulson, Assistant Computational Scientist

November 2022

1 Introduction to the Science problem

Next generation solid-state lithium batteries require the development of battery cathode materials with chemical and mechanical compatibility. We employ ensembles of machine learned force fields (MLFFs) to identify cathode atomic configurations and additives with the lowest energy and to evaluate their volume change with delithiation. Each training run for an individual MLFF is computationally expensive even on GPU resources. Accelerator hardware could dramatically improve training performance and enable simultaneous training of more MLFFs.

2 Description of the AI model and implementation

The design of battery cathode materials through computational means involves the exploration of massive configurational spaces. The number and size of atomistic calculations required are prohibitively computationally expensive and time consuming via traditional approaches such as density functional theory or molecular dynamics. Consequently, as part of a DOE VTO funding effort, the PIs have developed artificial intelligence (AI) driven workflows that dramatically reduce the computational burden [1]. Specifically, the team generates ensembles of DeePMDkit [2] MLFFs that are employed to screen for low energy configurations with various dopant concentrations and identities, and to predict the volume contraction with delithiation of those configurations selected. The training of hundreds of MLFFs and their evaluation for massive numbers of configurations quickly becomes computationally burdensome, even on cutting edge GPU supercomputers (approximately 36 hours on an A100 GPU on LCRC Swing per MLFF). The exploration of MLFF performance on ALCF testbed resources may lead to further dramatic reductions in the computational burden of cathode design. The PIs propose to examine the training of MLFFs on the ALCF Habana Gaudi processor due to its native support for Tensorflow, which underpins DeePMDkit models, and well-developed software stack. The proposed work is advantageous to Argonne for several compelling reasons: a) MLFFs are growing as a commonly used tool for production-level atomistic simulation, b) this would improve Argonne materials design capabilities, and c) the specific research application may lead to advancements enabling next-generation chemical energy storage solutions.

[1] Garcia, Juan C., Joshua Gabriel, Noah H. Paulson, John Low, Marius Stan, and Hakim Iddir. "Insights from Computational Studies on the Anisotropic Volume Change of Li x NiO2 at High States of Charge (x¡ 0.25)." The Journal of Physical Chemistry C 125, no. 49 (2021): 27130-27139.

[2] Wang, Han, Linfeng Zhang, Jiequn Han, and E. Weinan. "DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics." Computer Physics Communications 228 (2018): 178-184.

3 What was needed to get the model running on the AI Accelerator

PI-Paulson investigated in-depth the DeePMDkit open-source software. DeePMDkit is already optimized for scalable performance on multi-GPU systems. This is achieved via the CUDA Toolkit, mpi4py, and horovod. Integration with Lammps or I-Pi requires installation of a C++ interface with gcc ¿= 4.8. DeePMDkit is a large project, with 41 contributors and hundreds of thousands of lines of code. This complicated the task of adapting it to the Habana Gaudi system.

4 Performance Evaluation

The project did not make it to this stage.

5 Conclusion and next steps

The next steps would be to further investigate what would be required to work with mpi4py and horovod on the Habana Gaudi system. Then it would be possible to make the modification and test the code on MLFF training and prediction benchmarks versus the A100 processors currently employed.

Overall, progress towards the project goals was limited as the project team was unable to find expert help (via % effort) from those in CELS more familiar with adapting codes to alternative computing platforms. As such, all work was performed by PI-Paulson who had limited time to devote to the project.

PI-Paulson attended the Habana Gaudi workshop put on by Intel. Based on the workshop and associated documentation, Paulson gained access to the Habana Gaudi system, created a project directory, and successfully ran the provided Tensorflow test scripts.

Finally, the project team notes that motivation to adapt the DeePMDkit software to the Habana Gaudi 1 platform was limited, as it is outperformed by A100 GPUs by a considerable margin (according to benchmarks). Furthermore, the Gaudi 2 processor is not available at the Argonne testbed facility and would only slightly outperform the A100's even if available. As a team focused on results and performance, this makes development efforts less fruitful.

6 Acknowledgements

This research was supported by laboratory directed research and development funding from the U.S. Department of Energy through Argonne National Laboratory.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

Scalability Study of AI-based Surrogate for Ptychographic Image Reconstruction on Graphcore and Habana

Xiaodong Yu
Assistant Computer Scientist

Baixi Sun
Visiting Student
Indiana University

Zhen Xie
Postdoc Appointee

Tao Zhou
Assitant Scientist

Mathew Joseph Cherukara
Computational Scientist

October 2022

1 Introduction to the Science problem

Ptychography is a scanning Coherent diffraction imaging (CDI) technique that has been well-developed and widely used with optical, x-ray, and electron sources in scientific fields. It overcomes the resolution limits of typical x-ray optics by avoiding lens-imposed limitations of traditional microscopy and increasing the brilliance and coherence of the synchrotron light sources, hence can produce nanometer-scale resolution images of large volumes in thick samples with little sample preparation. Ptychographic image reconstruction is a phase retrieval problem, which is conventionally solved using model-based iterative methods that are computationally expensive [4]. Recently, a neural network-based solution, PtychoNN [2], has been proposed to replace the iterative methods. This surrogate learns a direct mapping from coherent diffraction patterns to real-space images and has been approved that it can achieve hundreds of time speedup than the traditional approaches on a single GPU with a small-scale training dataset. However, with the APS upgrade (APS-U) project, the data volume collected from the instrument will be dramatically expended [5], making the PtychoNN training cost an immense performance bottleneck. To address this issue, we move the training process onto the AI accelerators. In addition, the large-scale training dataset requires a large memory space usually beyond the capacity of a single device. Therefore, we need to understand the scalability of the AI accelerators and efficiently extend our PtychoNN training to multiple devices.

2 Description of the AI model and implementation

The neural network in PtychoNN is an encoder-decoder-based architecture, as shown in Figure 1. The input to the encoder is the diffraction data, and the output from the decoder is the real-space images. The two decoder arms yield the amplitude and phase parts of the images, respectively. The true labels for the decoder are obtained using the iterative phase-retrieval algorithm. The PtychoNN surrogate model was implemented using PyTorch with Distributed Data-Parallel (DDP) framework. This implementation required MPI (e.g., OpenMPI) and I/O libraries (e.g., HDF5) to support DDP in performing distributed data-parallel training on multiple devices. We used a training dataset containing 262,896 images in total (diffraction, phase, and amplitude), and the size of each image was 128x128 (65KB). The dataset was split into training, validation, and testing subsets following the 8:1:1 ratio.

3 What was needed to get the model running on the AI Accelerator

The initial plan described in the proposal is to benchmark the scalability of PtychoNN on Graphcore devices. However, during the deployment, we encountered an MPI issue with Graphcore that we could not overcome,

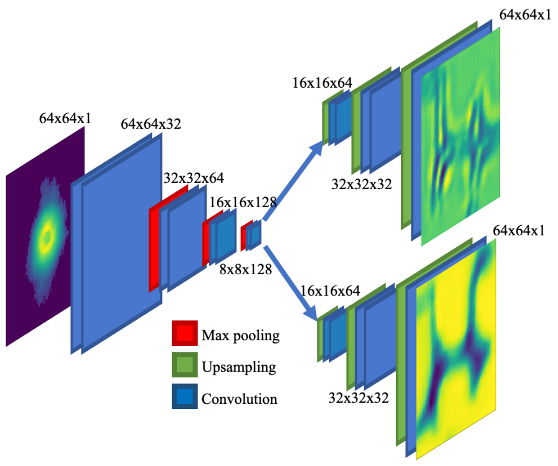
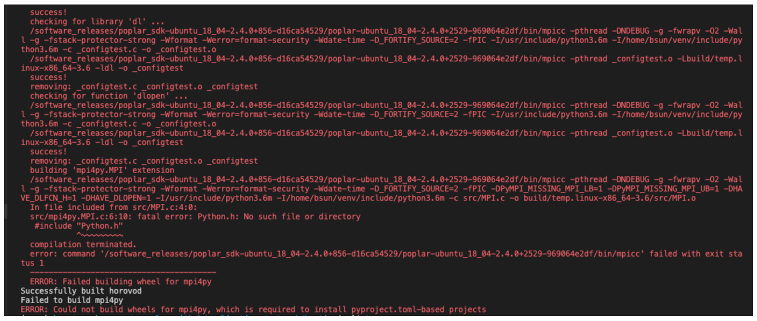


Figure 1: The network architecture of PtychoNN [2].

which was later confirmed as a driver issue by the engineers from the vendor. Given that this issue cannot be solved until the latest model of Graphcore is deployed in Argonne, after consulting with the ALCF staff, we decided to switch to the Habana-based platform. We have successfully run PtychoNN on Habana and tested the scalability. Below are the deployment details on both Graphcore MK1 and Habana Gaudi 1:

Graphcore We first tried to deploy our Pytorch-based PtychoNN on a single Graphcore. To this end, we first need to load the Popart and Poplar environments, then build our own virtual environment by installing the packages. After the virtual environment was built, we were able to run PtychoNN on a single Graphcore and observed about 20% training time improvement to that on an NVIDIA A100 GPU.

We then tried to scale the PtychoNN training onto multiple Graphcore devices. For this purpose, we need to enable the PyTorch DDP framework in the environment of the Graphcore platform. Unfortunately, the default MPI in the python virtual environment was not compatible with PyTorch DDP. Specifically, there was an error with the message *TypeError: can't pickle _thread.lock objects*. The engineer from Graphcore suggested that we possibly can avoid this issue by building our own MPI. Therefore, we tried to build mpi4py in our python environment. But it failed with the *Cannot build the wheels for mpi4py* error as shown in Fig. 2. We worked with the Graphcore engineers for several weeks and finally sorted out that it was a driver issue and would not be solved for the old Graphcore MK1 model in the current ALCF AI testbed. We were notified the latest Graphcore MK2000 model would only be available on the testbed someday after the LDRD expedition end date, hence we decided to move to the Habana Gaudi platform according to the suggestions from ALCF staff.



```
  success!
  checking for library 'dl' ...
  /software_releases/poplar_sdk-ubuntu_18_04-2.4.0+856-d16ca54529/poplar-ubuntu_18_04-2.4.0+2529-969064e2df/bin/mpicc -pthread -DNDEBUG -g -fwrapv -O2 -Wal
l -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2 -fPIC -I/usr/include/python3.6m -I/home/bsun/venv/include/py
thon3.6m -c _configtest.c -o _configtest.o
  /software_releases/poplar_sdk-ubuntu_18_04-2.4.0+856-d16ca54529/poplar-ubuntu_18_04-2.4.0+2529-969064e2df/bin/mpicc -pthread _configtest.o -Lbuild/temp.l
inux-x86_64-3.6 -ldl -o _configtest
  success!
  removing: _configtest.c _configtest.o _configtest
  checking for function 'dlopen' ...
  /software_releases/poplar_sdk-ubuntu_18_04-2.4.0+856-d16ca54529/poplar-ubuntu_18_04-2.4.0+2529-969064e2df/bin/mpicc -pthread -DNDEBUG -g -fwrapv -O2 -Wal
l -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2 -fPIC -I/usr/include/python3.6m -I/home/bsun/venv/include/py
thon3.6m -c _configtest.c -o _configtest.o
  /software_releases/poplar_sdk-ubuntu_18_04-2.4.0+856-d16ca54529/poplar-ubuntu_18_04-2.4.0+2529-969064e2df/bin/mpicc -pthread _configtest.o -Lbuild/temp.l
inux-x86_64-3.6 -ldl -o _configtest
  success!
  removing: _configtest.c _configtest.o _configtest
  building 'mpi4py.MPI' extension
  /software_releases/poplar_sdk-ubuntu_18_04-2.4.0+856-d16ca54529/poplar-ubuntu_18_04-2.4.0+2529-969064e2df/bin/mpicc -pthread -DNDEBUG -g -fwrapv -O2 -Wal
l -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2 -fPIC -DPyMPI_MISSING_MPI_LB=1 -DPyMPI_MISSING_MPI_UB=1 -DHA
VE_DLFCN_H=1 -DHAVE_DLOPEN=1 -I/usr/include/python3.6m -I/home/bsun/venv/include/python3.6m -c src/MPI.c -o build/temp.linux-x86_64-3.6/src/MPI.o
  In file included from src/MPI.c:4:0:
  src/mpi4py.MPI.c:6:10: fatal error: Python.h: No such file or directory
   #include "Python.h"
            ^~~~~~~~~~
  compilation terminated.
  error: command '/software_releases/poplar_sdk-ubuntu_18_04-2.4.0+856-d16ca54529/poplar-ubuntu_18_04-2.4.0+2529-969064e2df/bin/mpicc' failed with exit sta
tus 1
  ----------------------------------------
  ERROR: Failed building wheel for mpi4py
Successfully built horovod
Failed to build mpi4py
ERROR: Could not build wheels for mpi4py, which is required to install pyproject.toml-based projects
```

Figure 2: The screenshot of the mpi4py building issue in the environment of the Graphcore MK1 platform.

Habana In order to test the scalability, we need to deploy PtychoNN on Habana Gaudi with PyTorch DDP enabled. Following the user guide instructions, we first created a virtual environment and installed the site-packages using the command *python3 -m venv –system-site-packages path_to_env* to set the environment variables. In particular, we defined the *HABANA_LOGS* value to specify the output directory of log files.

The information in the logs was crucial for profiling the system performance, for example, understanding the HPU computing resources and memory utilization. It was also noteworthy that the python path should be manually added using *export PYTHONPATH=path_to_env/bin/python3* after the virtual environment was activated by *source path_to_env/bin/activate*.

During the scalability study, we observed the following issues: (i) The profiling tool provided by PyTorch failed to monitor the HPU activities. It only reported the details of CPU activities, and we had to use the system timer to measure the HPU performance. (ii) We had to manually specify the public keys of all nodes in each node's *known_hosts*, in order to run the cross-node training with Habana. (iii) The h5py library cannot be successfully built on both Habana nodes. We had to use the traditional numpy load to work around it.

4 Performance Evaluation

In this section, we will present the scalability and accuracy results of PtychoNN majorly on Habana Gaudi machines. The AI testbed had two Habana nodes with 8 HPUs residing on each node and thus had 16 HPUs in total. All experiments were conducted with a 22GB real training dataset collected at the beamline.

In the single-device context, we observed that Graphcore achieved the best training performance for PtychoNN. Specifically, It outperformed NVIDIA A100 GPU by 1.2-fold [3], while the A100 GPU can be 2× faster than Habana Gaudi 1. We note that Habana Gaudi 1, which was on the current AI testbed, was the old model designed as a competitor to NVIDIA V100 GPU and was reported by its vendor as showing worse peak performance than A100 GPU [1]. The next-generation model, Habana Gaudi 2, would have much better performance and was expected to outperform A100 GPU.

We then evaluated the training scalability of PtychoNN on different machines. As described in Sec. 3, we failed to scale Graphcore-based implementation to multiple devices due to driver issues; therefore, we only presented here the scalability comparisons between Habana Gaudi 1 and A100 GPUs. We performed both strong and weak scaling studies. There were three parameters to scale: *per device batch size* (*pdbs*), *device number* (*dnum*), and *global batch size* (*gbs*) = *pdbs* × *dnum*. *pdbs* defined the per-device workload during each training step, while *gbs* determined the number of steps in the entire training. In weak scaling, we fixed *pdbs* at 64 and changed *dnum* from 1 HPU (GPU) to 2, 4, 8, and 16 HPUs (GPUs). Accordingly, *gbs* was changed from 64 to 128, 256, 512, and 1024. We noted that with the same training data size, larger *gbs* led to fewer total training steps; hence the end-to-end training time could be reduced. Fig. 3 showed weak scaling results on both Habana Gaudi 1 and A100 GPU. We can observe that although 2 HPUs took a slightly longer time than 1 HPU, it exhibited near-linear scalability from 2 to 8 HPUs. However, we also observed that 16 HPUs performed worse than 8 HPUs. This could be because the cross-node communication for Habana was not well optimized.

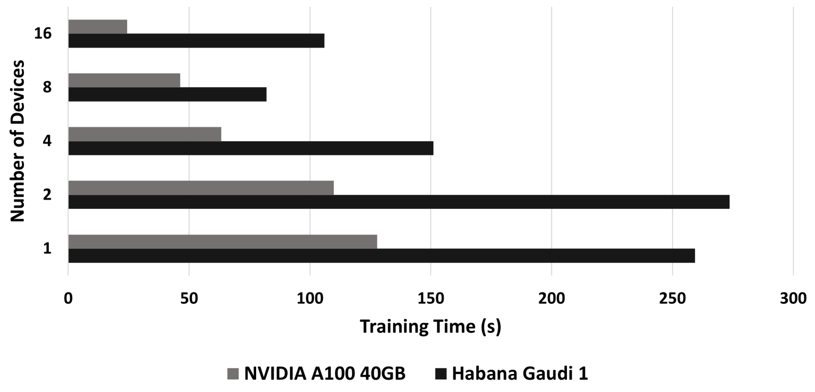


Figure 3: Weak Scaling results on both Habana Gaudi 1 and A100 GPUs. *pdbs* was fixed at 64.

In the strong scaling context, we fixed *gbs* at 512 and scaled up *dnum* from 1 to 8 HPUs. Accordingly, *pdbs* scaled from 512 down to 64. Fig. 4 depicted the strong scaling results on Habana. From the figure we observed that the strong scaling was sub-linear. This was reasonable because the HPU-HPU communications had overhead.

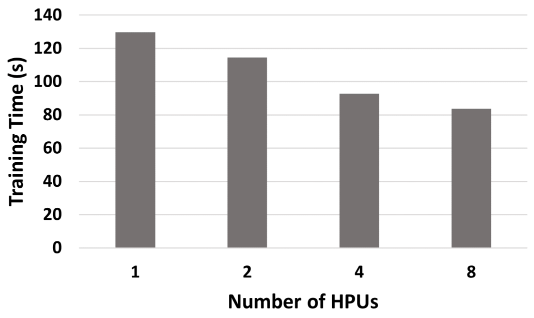


Figure 4: Strong Scaling results on Habana Gaudi 1. *gbs* was fixed at 512.

To make the study complete, we also fixed *dnum* at 8 and scaled up *pdbs* from 32 to 256. Therefore, *gbs* was changed from 256 to 2048. Fig. 5 showed the scalability results. We can see that, along with the *pdbs* increase, the training time was reduced.

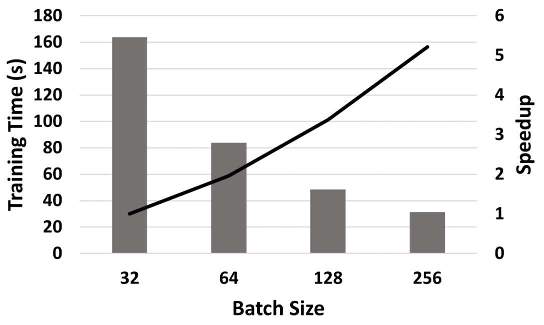


Figure 5: Scalability results with fixed *dnum* at 8 HPUs. The bars indicate the per epoch training time, and the line shows the speedups compared to that of $pdbs = 32$.

We finally evaluated the validation loss on Habana Gaudi 1 and compared it to that on A100 GPUs. Fig. 6 presented the validation loss after each epoch on Habana. We can observe that PtychoNN converged after around 70 epochs and reached a loss below 0.2, which was within the acceptable range. We also compared the validation loss between HPUs and A100 GPUs with different *dnum*, as shown in Fig. 7. We highlighted that with 8 and 16 devices, the validation losses on Habana were significantly lower than that on GPUs. The scaling out on GPUs drastically increased the validation loss while scaling out on HPUs will not. We had yet to figure out the reasons for this phenomenon.

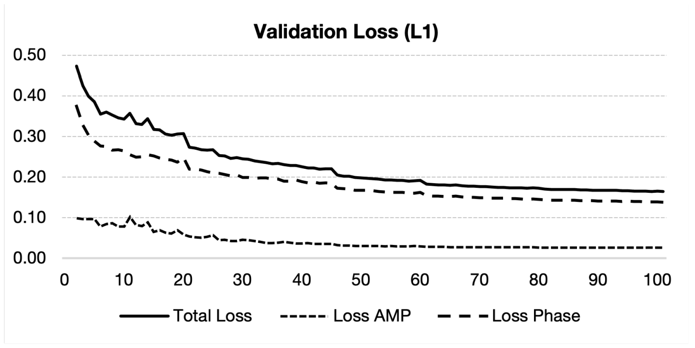


Figure 6: Validation loss w.r.t. epochs. It indicates the convergence speed on Habana.

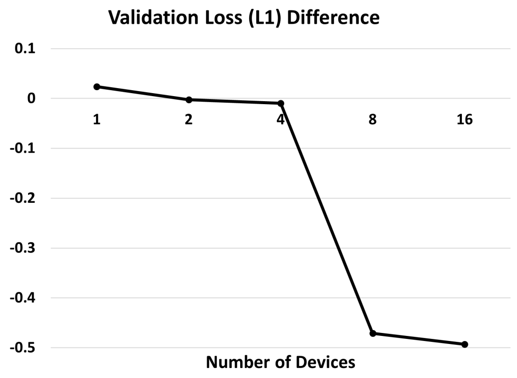


Figure 7: Validation loss comparisons between A100 GPU and Habana. A negative value indicates Habana has a lower loss than A100. The lower the loss, the better accuracy.

5 Conclusion and next steps

In this work, we studied the scalability of the PtychoNN surrogate on AI accelerators. We first chose the Graphcore platform and successfully deployed PtychoNN on a single Graphcore machine. However, we were stuck by a driver issue to scale our training onto multiple Graphcore machines. This issue cannot be solved until the next-generation Graphcore, MK2000, would be installed. Therefore, we switched our focus to the Habana Gaudi platform. We successfully deployed PtychoNN on Habana and scaled out the training to multiple HPUs on multiple nodes. The experiments showed a good strong scaling result and a good weak scaling result from 2 to 8 HPUs. Training on 16 HPUs performed worse than that on 8 HPUs, possibly because the cross-node communication was not well optimized for Habana. The training performance on current Habana Gaudi 1 machine was worse than that on the NVIDIA A100 GPU due to the hardware limit; we expected the performance would be drastically improved with the next-generation Habana Gaudi 2. We also evaluated the validation loss and observed that PtychoNN can achieve higher accuracy on Habana than on A100 with a good convergence speed.

For the next steps, we will complete the scalability study on Graphcore after the new model is installed. We will get more insight into the scalability issue we observed on Habana Gaudi and try to solve them. We also plan to study the scalability of other accelerators on the AI testbed.

6 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] Second-Gen Habana Gaudi2 Outperforms Nvidia A100. `https://www.intel.com/content/www/us/en/newsroom/news/second-gen-habana-gaudi2-outperforms-nvidia-a100.html#gs.em9y3p`. (Accessed on 10/10/2022).

[2] Mathew J Cherukara, Tao Zhou, Youssef Nashed, Pablo Enfedaque, Alex Hexemer, Ross J Harder, and Martin V Holt. AI-enabled high-resolution scanning coherent diffraction imaging. *Applied Physics Letters*, 117(4):044103, 2020.

[3] Baixi Sun, Xiaodong Yu, Kamil Iskra, and Dingwen Tao. SurrogateTrain: Drastically Improving Performance of Data Loading for Training Scientific Surrogate Models. In *the 2022 International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2022. (Poster).

[4] Xiaodong Yu, Viktor Nikitin, Daniel J Ching, Selin Aslan, Doğa Gürsoy, and Tekin Biçer. Scalable and accurate multi-GPU-based image reconstruction of large-scale ptychography data. *Scientific Reports*, 12(1):1–16, 2022.

[5] Ke Yue, Junjing Deng, Yi Jiang, Youssef Nashed, David Vine, and Stefan Vogt. Ptychopy: GPU framework for ptychographic data analysis. In *X-Ray Nanoimaging: Instruments and Methods V*, volume 11839, page 118390F. International Society for Optics and Photonics, 2021.

Computing, Environment and Life Sciences
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC