Accelerating CPU-based Distributed DNN Training on Modern HPC Clusters using BlueField-2 DPUs

Arpan Jain, Nawras Alnaasan, Aamir Shafi, Hari Subramoni, and Dhabaleswar K Panda
Department of Computer Science and Engineering,
The Ohio State University, Columbus, Ohio, USA,
{jain.575, alnaasan.1, shafi.16, subramoni.1}@osu.edu, panda@cse.ohio-state.edu

***Abstract*—The Deep Learning (DL) training process consists of multiple phases — data augmentation, training, and validation of the trained model. Traditionally, these phases are executed either on the CPUs or GPUs in a serial fashion due to lack of additional computing resources to offload independent phases of DL training. Recently, Mellanox/NVIDIA has introduced the BlueField-2 DPUs which combine the advanced capabilities of traditional ASIC based network adapters with an array of ARM processors. In this paper, we characterize and explore how one can take advantage of the additional ARM cores on the BlueField-2 DPUs to intelligently accelerate different phases of DL training. We propose multiple novel designs to efficiently offload the phases of DL training to the DPUs. We evaluate our proposed designs using multiple DL models on state-of-the-art HPC clusters. Our experimental results show that the proposed designs are able to deliver up to 15% improvement in overall DL training time. To the best of our knowledge, this is the first work to explore the use of DPUs to accelerate DL training.**

***Index Terms*—DNN Training, DPU, CPU, Offloading, PyTorch, ARM, Horovod, MPI**

I. Introduction

The hyperscale data centers have been using Smart NICs to offload a variety of functions from the host processor. These functions typically include data and control plane switching, Network Function Virtualization (NFV), intrusion detection, encryption, and compression. This trend is effective since it relieves the host processor cores to focus entirely on user workloads and applications leading to better return on investment.

More recently, Smart NICs (or Intelligent NICs) like BlueField-2 Data Processing Units (DPUs) have been introduced in the HPC community to enable offload of high-value communication and compute operations. Unlike the data center community, the exploration of these DPUs as additional processing elements—along with CPUs and GPUs—is still in its infancy. One approach [1] is to offload MPI communication functionality on DPUs. While this is promising, it is also possible to offload subsets of computation on DPUs in addition to communication functions. We explore this approach in the context of Deep Learning (DL) workloads in this paper.

Distributed Deep Learning has become the default approach to achieve models with high accuracy in areas like natural language processing, computer vision, and recommendation systems. DL uses deep neural networks (DNNs) to learn the relationship between input and output by training it on a large corpus of data. However, training DL models is a compute-intensive and time-consuming task as it can take weeks or months. Therefore, state-of-the-art DL models like AmoebaNet [2] and GPT-3 are trained on multiple computing nodes using distributed DNN training. In recent years, Intel has optimized its processors for DNN training using MKL-DNN library. Hence, CPU-based DNN training is gaining a lot of traction in the community. In this paper, we explore the possibility of offloading different phases of DL training to DPUs.

*This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, and XRAC grant #NCR-130002.

II. Contributions

In this paper, we first characterize the DNN training on a single node for both CPU and DPU. We then optimize various parameters like *OMP_NUM_THREADS*, number of processes per node, and batch size for PyTorch DL framework to get good performance on CPU and DPU. We use Multiple Process Multiple Data (MPMD) support of MPI to run DL training on heterogeneous architectures (Intel and ARM cores). Based on our characterization, we explore the possibility of offloading different DNN training phases to DPUs to accelerate CPU-based DNN training on multi-node heterogeneous system comprising a set of CPUs and DPUs. We present performance evaluation results on 16 nodes, each with 32 CPU cores and 8 ARM cores. To the best of our knowledge, this is the first work to explore the use of DPUs to accelerate DL training on a multi-node heterogeneous CPU+DPU cluster.

To summarize, this paper makes the following contributions:

- Performance characterization of Data Parallelism on CPU and DPU using multi-process per node configuration
- Proposing multiple novel designs for offloading different phases of DL training to the BlueField-2 DPUs
- Performance evaluations of the proposed designs with multiple DL models (ResNet-20, ResNet-56, and ShuffleNet) on three datasets.
- Obtained speedup improvements up to 15%, 12.5%, and 11.2% for training the ResNet-20 model on the CIFAR-10 dataset, ShuffleNet model on the Tiny ImageNet dataset, and ResNet-56 model on the SVHN dataset respectively.
- Scaled proposed designs to 16 nodes and achieved consistent improvement for multi-node experiments.

III. Background

A. Deep Neural Network (DNN) Training

Deep Neural Network (DNN) can be viewed as a mathematical function with learnable parameters that transform the input into output. It learns the relationship between input and output data by updating the parameters known as weights. DNN is composed of several smaller mathematical functions known as neurons that combine weighted summation and activation function. Weighted summation is a linear function; hence activation function is required to introduce nonlinearity into DNN. *Layers* are formed by grouping the subsequent sets of neurons. DNN training has two steps: 1) Forward Pass and 2) Backward Pass. In the forward pass, the output is calculated by feeding the input to the first layer of DNN and recursively applying the next layer to the previous layer's output. The output of DNN is compared with the actual output to calculate the *Loss function* and errors. Backward pass back propagates the error to every layer to calculate the gradients used to update the weights of DNN. This process is repeated several times to reduce the loss function and get better accuracy.

B. Deep Learning Frameworks

DL frameworks provide a high-level interface to develop, train, and test DL models for any application area like computer vision, speech recognition, and natural language processing. DL frameworks are optimized for various CPU, GPU, TPU, and AI-specific architectures. DL frameworks provide basic building blocks to customize the DNN according to the application. They hide most of the complicated mathematics like differentiation in the backward pass and provide an easy-to-use interface to train DL models.

C. Distributed DNN Training

Distributed DNN training can be categorized into three approaches; 1) Data Parallelism, 2) Model Parallelism, and 3) Hybrid Parallelism. Data parallelism creates a model replica on each processing element and conducts forward and backward pass simultaneously. At the end of the backward pass, the model is synchronized using an allreduce operation. In model parallelism, a model is distributed over multiple processing elements. Distributed forward and backward pass is implemented to train the model [3], [4]. Hybrid parallelism [3], [5] combines model-parallelism with data parallelism to scale the distributed training to a large number of GPUs.

D. BlueField-2 DPU

Data processing units (DPUs) are rapidly becoming more prominent in supercomputing for various purposes. They mainly come with a wide range of network interfaces and are used to enable resources on the server to move data more efficiently. The BlueField-2 DPU is a system-on-chip architecture which comes with ARMv8 A72 cores (64-bit), DDR4 DRAM controller, ethernet, InfiniBand, and PCI interfaces. BlueField-2 DPUs are used to offload I/O intensive tasks from CPU taking the role of a traffic controller with greater performance. Furthermore, BlueField-2 DPUs may be used to offload encryption and compression workload from the CPU [6].

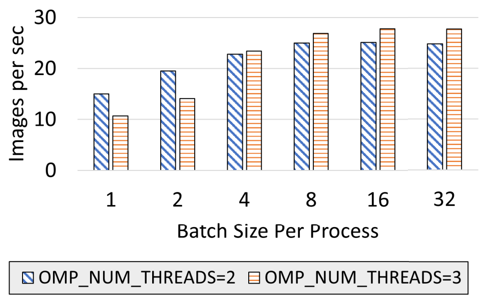
IV. Exploiting DPUs for Deep Neural Network Training

Deep Neural Network training consists of several phases like fetching training data, data augmentation, forward pass, backward pass, weight update, and model validation. These phases can be offloaded to Bluefield-2 DPUs to accelerate the DNN training. In this section, we discuss different ways to accelerate the training by offloading certain tasks to DPU.

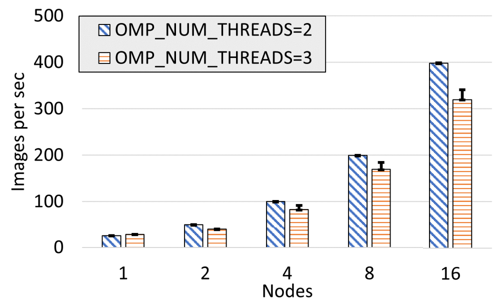
A. Offload Naive (O-N): Offloading DL Training using Data Parallelism

There are several ways to distribute DNN training on multiple processing elements. We use data parallelism to offload training to DPUs as it has low communication overhead and data dependency among different processing elements. Data parallelism creates a model replica on each processing unit to perform forward and backward pass simultaneously. Allreduce operation is used to synchronize the gradients after backward pass. Since CPU and DPU differ in the number of cores, frequency, and computation power, we first characterize the DL training on a single CPU and DPU, then we use this characterization to distribute work between CPU and DPU.

CPU Performance Evaluation: Recent characterization studies [7], [8] have suggested multiple processes per CPU to get the best performance for DL training. Each process creates a model replica and uses data parallelism to perform DL training. Therefore, we conduct multi-process experiments on single CPU to find the sweet spot between batch size, OMP_NUM_THREADS, and the number of processes per node (PPN). Based on our evaluations, we set PPN to 8 for PyTorch. Figure 1(a) shows the performance for different batch sizes and OMP_NUM_THREADS. The performance saturates after batch size of 16 per process. Therefore, we use batch size of 16 per process in our next set of evaluations. In our multi-node experiments, we found that performance varies for OMP_NUM_THREADS=3 as PyTorch initiates two extra active threads in addition to the given value. Figure 1(b) shows the performance evaluation for multi-node experiments. Based on our characterization, we choose batch size of 16 per process and OMP_NUM_THREADS=2 for CPU+DPU experiments.
DPU Performance Evaluation: We perform similar multi-process experiments for DPUs to find the best configuration. Based on our experiments, we choose 2 PPN for DPUs. Figure 2 shows the performance evaluation for different batch sizes and OMP_NUM_THREADS. Since DPUs have 8 cores, we limit the number of threads to 3 as one thread is used by Horovod for communication.
Offload Naive Performance Evaluation: In order to offload training to DPU, we create additional model replicas on DPU using data parallelism and distribute work between CPU and DPU by specifying different batch sizes per process for CPU and DPU. Figure 3 shows an example of data parallelism on



(a) Single Node



(b) Multi-Node

Fig. 1. Performance Characterization of ResNet-50 on CPU

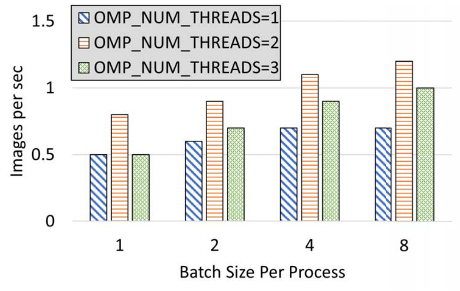


Fig. 2. Performance Characterization of ResNet-50 on DPU

CPU and DPU for 8 PPN per CPU and 2 PPN per DPU. We select 16 batch size per process for CPUs but we cannot select 8 batch size per process for DPU as the time per batch should be the same for both CPU and DPU. Based on time per batch, we found that 2 batch size per process on DPU takes approximately same time as 16 batch size per process for CPUs. Therefore, we use 16 batch size per process for CPU and 2 batch size per DPU for Offload Naive experiments.

Figure 4 shows the comparative performance of CPU and Offload Naive for the ResNet-50 model. We report up to 1.03× speedup using the proposed Offload Naive design. For training, the maximum speedup possible is 1.04X as the individual performance for CPU and DPU is 25.1 and 1, respectively. Forward and Backward passes are compute-intensive tasks and DPUs are not as powerful as CPUs. Therefore, offloading training to DPUs will not give significant speedup. Hence, we evaluate the possibility of offloading other phases of DNN training to DPUs in the next section.

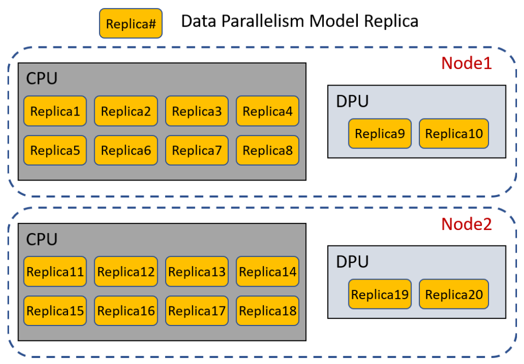


Fig. 3. Model replica placement for Offload Naive experiment in data parallelism

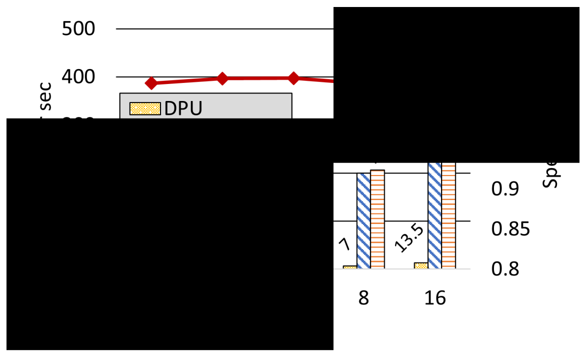

[redacted]
[redacted]

Fig. 4. Evaluation of proposed Offload Naive design

B. *Offloading Data Augmentation and Model Validation*

DL training consists of several phases and steps. Figure 5 shows different phases in DNN training. Each epoch consists of fetching training data, data augmentation, forward pass, backward pass, weight update, and model validation. Forward and backward passes are the most compute-intensive operations. However, other operations also contribute to the overall time per epoch. In this section, we explore the possibility of offloading these operations to DPUs and characterize the time taken by these operations.

Data augmentation is a set of user-defined operations that are applied to raw training data before forward pass. Data augmentation helps in generalizing the training of DNN by making minor alterations to the input training data. Normalization, resizing, ZCA whitening, random rotation, random zoom, and random flip are standard data augmentation functions used in computer vision. In model validation, loss and accuracy are calculated for validation/testing data to avoid overfitting and calculate the performance of the DL model after every epoch. Size of validation/testing data and operations in data augmentation depends on DNN, dataset, and DL researcher. Therefore, we calculate the time for data augmentation (includes the reading of training data), model validation, and forward/backward pass time for different models and datasets. Figure 6 shows the time spent on data augmentation, model validation, and forward/backward pass. Data augmentation and model validation time range from 4% to 12% and 2% to 10%

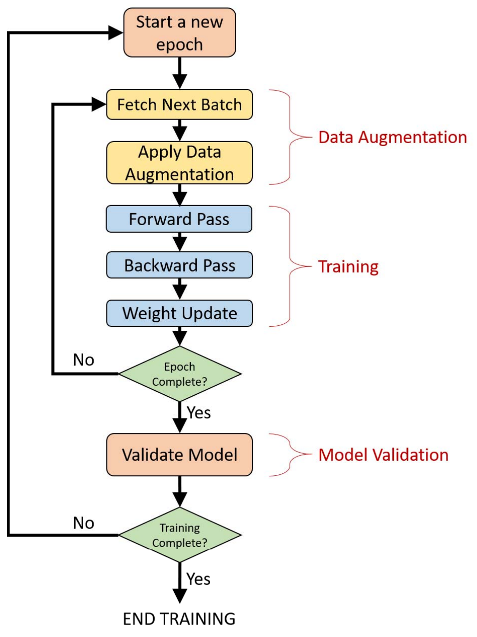


Fig. 5. Phases in DNN training

of the overall time per epoch, respectively. Therefore, data augmentation and model validation can be offloaded to DPUs to achieve better speedup for CPU+DPU.

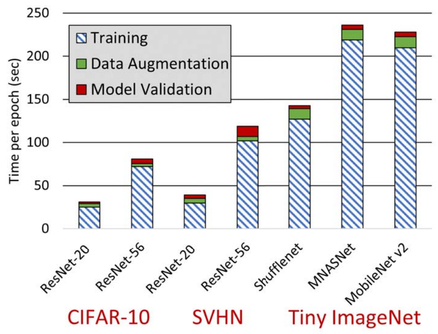


Fig. 6. Time spent on data augmentation (includes training data reading time), model validation, and forward/backward pass

V. Proposed Advance Offloading Designs

We investigate several designs to offload different phases of DNN training to DPUs and reduce the time per epoch. We propose strategies to offload data augmentation and model validation to DPUs and overlap these phases with forward and backward pass on CPUs. We use MPI4py for the communication between processes on CPUs and DPUs. Horovod is used to train the model with data parallelism on CPUs. Overall we propose three designs: 1) Offloading data augmentation, 2) Offloading model validation, and 3) Hybrid offloading that combines data augmentation and model validation.

A. Design 1: Offload Data Augmentation (O-DA)

In this design, we offload the reading of training data from memory and data augmentation on input data to DPUs. Figure 7 shows the overall proposed design. There are two types of processes: 1) Training process and 2) Data augmentation process. For each training process on CPU, we initialize a data augmentation process on DPUs. Training process on CPU does forward pass, backward pass, gradient synchronization (for data parallelism), weight update, and model validation steps. Data augmentation process on DPUs fetch the training data from storage, apply user-defined data augmentation functions, and send the batch of input and output to training process on CPU. Since DPUs have eight cores, we limit the number of processes per CPU/DPU to 8.

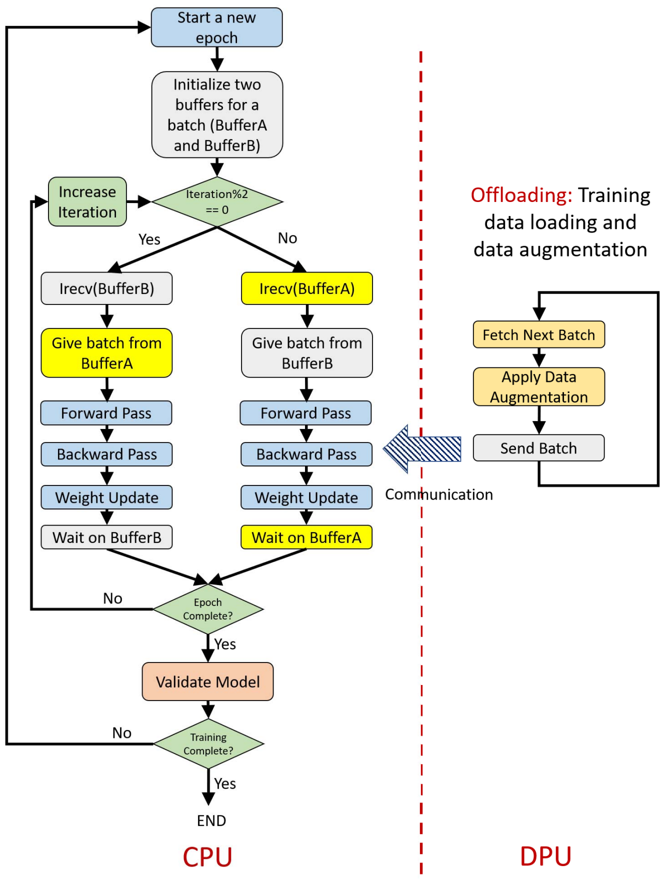


Fig. 7. Flow chart for proposed Offload Data Augmentation (O-DA) design. It offloads the reading of data from memory and data augmentation functions to DPU.

Training process: We create two buffers of the same size as input and output to overlap commutation overhead with forward and backward pass of DNN training on CPUs. One buffer is used to receive the next batch from the data augmentation process and another buffer is used to perform forward and backward pass. An irecv operation is initialized before forward pass to overlap communication with computation.

Data augmentation process: We divide the training data among processes on DPUs using Pytorch's "DistributedSampler" class. Each data augmentation process initializes a set of circular buffers to overlap communication with computation on DPUs. If a free buffer is available, it fetches the next batch, applies data augmentation functions, and posts an isend to the training process. Otherwise, it waits for a buffer to become available.

B. Design 2: Offload Model Validation (O-MV)

Instead of offloading data augmentation to DPU, we offload model validation to the DPUs. Model validation is a less compute-intensive task compared to training. It is similar to inference using a trained model. We overlap the calculation of validation loss and accuracy for epoch i with the training of epoch $i+1$. Figure 8 shows the offloading of model validation to DPU for one CPU and DPU process. We create two types of processes in this design: 1) Training Process on CPU and 2) Testing process on DPU.

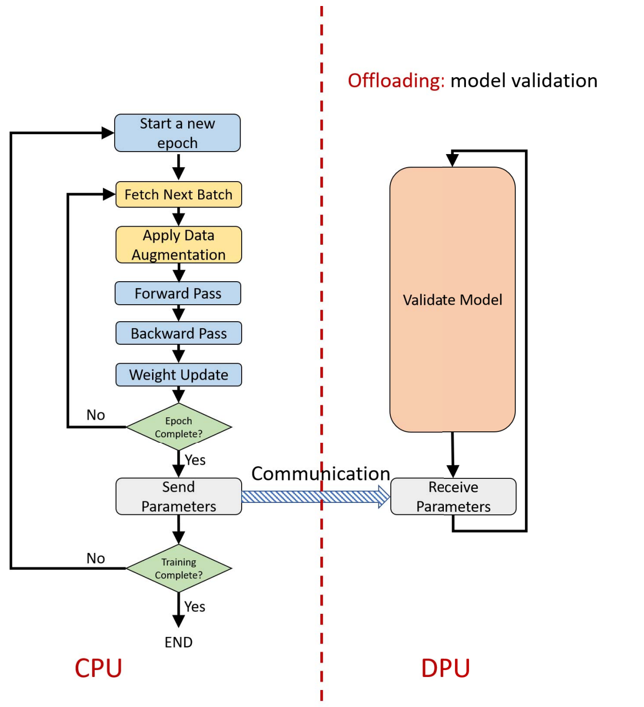


Fig. 8. Flow chart for proposed Offload Model Validation (O-MV) design. It offloads the validation of model on testing dataset and overlaps the computation with the training of next epoch.

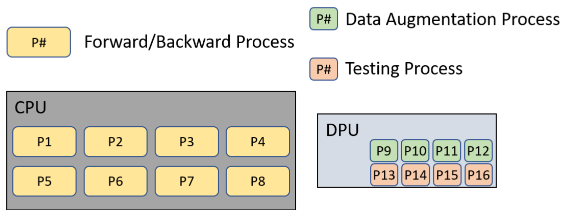
Training process: In this strategy, the training process fetches the training data from memory, applies data augmentation, and performs forward pass, backward pass, and weight update. After performing training on all the training samples, the training process sends parameters to the corresponding DPU process using point-to-point communication primitives. It moves to the next epoch after sending weights to the testing process.

Testing process: For every training process on the CPU, a testing process is initialized on DPUs. The testing process waits for the weights from the corresponding training process for epoch i. Since DPUs are slower than CPUs, we expect the model validation part on DPUs to take equal or less time than the training part on CPUs.

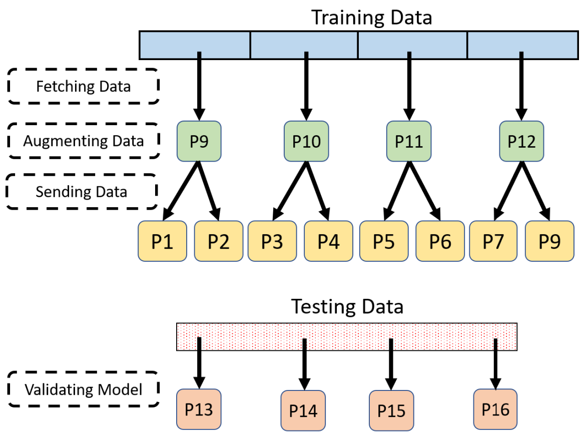
In our evaluation, we found that model validation on DPUs may take more time than the training part on CPUs for some models and datasets. This leads to degradation as time per epoch is the maximum of the testing part time on DPU and the training part time on CPUs. Therefore, for such cases, we do not offload the model validation to DPUs completely. We divide the testing data between CPU and DPU to balance the total time and achieve good overlap.

C. Design 3: Offload Hybrid (O-Hy)

We combine offloading of data augmentation and model validation to achieve better speedup for the model that spent a significant amount of time in forward and backward pass (training). We create three types of processes: 1) Forward/backward process on CPU, 2) Data augmentation process on DPU, and 3) Testing process on DPU. Figure 9(a) shows the process placement of all three types of processes. Since CPU has 32 cores, each process runs on four cores. Likewise, each DPU process runs on one core. Figure 9(b) shows the high-level distribution of work based on the type of data among processes. Figure 10 shows the flow diagram of hybrid offloading to DPU.



(a) Process placement on CPU and DPU



(b) Data movement for training and testing data

Fig. 9. Different types of processes in hybrid offloading that offloads data augmentation and model validation to DPU and coordinates DNN training CPU.

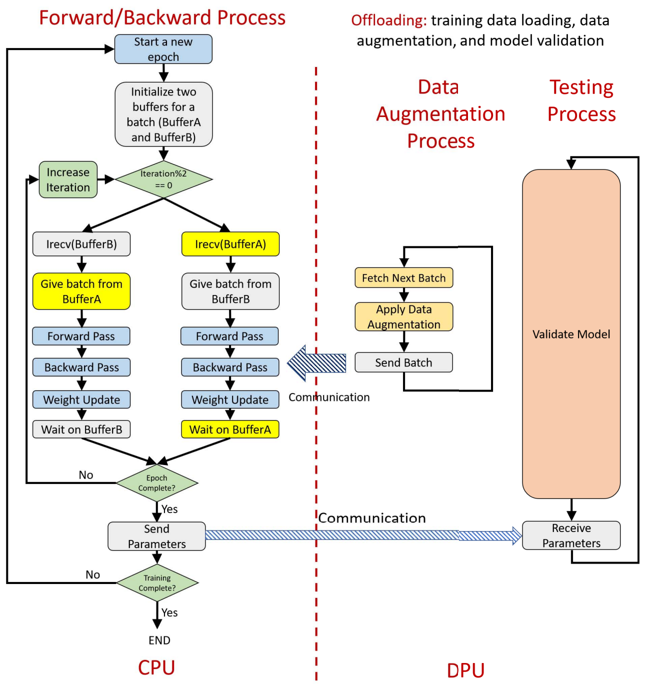


Fig. 10. O-Hy: Offload Hybrid (data augmentation and model validation) to DPU.

Forward/Backward Process: Processes running on CPUs are called forward/backward processes in this design. They receive augmented training data from the data augmentation process on DPU and perform forward pass, backward pass, and weight update for the given training batch. At the end of the training epoch, it sends weights of DNN to a testing process on DPU. **Data Augmentation Process:** We run four data augmentation process on each DPU that fetch training data from memory, augment the data, and send it to the forward/backward process on CPU. We use the same optimizations for the data augmentation process discussed in Section V-A. In hybrid design, each data augmentation process sends data to two forward/backward processes on the CPU using asynchronous communication. **Testing Process:** Testing processes validate the DL model on the validation/testing dataset. It overlaps the validation with the training of the next epoch. We divide testing data among testing processes and use the same optimizations discussed in Section V-B.

VI. Evaluating Performance of Proposed Offloading Approaches

This section provides a comprehensive performance evaluation of our proposed offloading designs using a variety of DL models and datasets. These proposed approaches include Design #1: Offload Data Augmentation (**O-DA**), Design #2: Offload Model Validation (**O-MV**), and Design #3: Offload Hybrid (**O-Hy**). Note that **O-Hy** combines **O-DA** and **O-MV**. We also add **No Offload** to our evaluation in order to provide baseline performance in the absence of DPUs.

A. *Experimental Setup*

We used the HPC Advisory Council High-Performance Center (HPCAC) [9] cluster for our evaluation. HPCAC has 32 nodes that contain the BlueField-2 network adapters. These adapters have an array of 8 ARM cores operating at 1999 MHz with 16 GB RAM. Each BlueField-2 adapter is equipped with Mellanox MT41682 EDR ConnectX-5 HCAs (100 Gbps data rate) with PCI-Ex Gen3 interfaces [6]. The host is equipped with the Broadwell series of Xeon dual-socket, 16-core processors operating at 2.60 GHz with 128 GB RAM.

For the purpose of evaluating DL training workloads, this section used the following models: ResNet-20 [10], ResNet-56 [10], and ShuffleNet [11], The following datasets were used: CIFAR-10 [12], [13], Street View House Numbers (SVHN) [14] Dataset, and Tiny ImageNet [15].

Software Libraries: PyTorch v1.9 [16], Horovod v0.21 [17], MPI4py v3.0.3 [18], and MVAPICH2 2.3.6 MPI library [19]

B. *Single Node Experiments*

This sub-section presents performance evaluation of our proposed designs using various DL models and datasets on a single node of the HPCAC system. The idea here is to understand the performance on one node before scaling our proposed designs to multiple nodes in Section VI-C. As described earlier in Section VI-A, a single node is equipped with 32 Intel CPU cores and 8 ARM DPU cores. We launch 8 MPI processes on the host processor—each MPI process has 4 OpenMP threads in order to fully utilize the available 32 processing cores on the system. Also, we launch 8 additional MPI processes on the BlueField-2 DPU to fully exploit the avaiable 8 ARM cores. These 8 MPI processes are used for different purposes in the proposed designs—data augmentation in **O-DA**, model validation in **O-MV**, and data augmentation and model validation in **O-Hy**. For the **O-Hy** design, 4 MPI processes are used for both data augmentation and model validation in a hybrid manner.

Training ResNet-20 with the CIFAR-10 Dataset: Figure 11 depicts the time taken for each epoch during training for no offload and proposed designs. **O-DA** and **O-MV**, achieve 13.8% and 3.1% speedups, respectively. In this experiment, the **O-Hy** design results in a slow down. In this case, performing data augmentation on DPU cores produced best overall results.

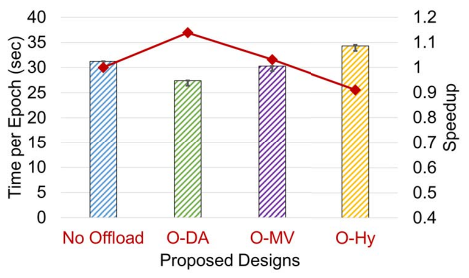


Fig. 11. Performance characterization of training ResNet-20 model with CIFAR-10 dataset using different designs

Training ResNet-56 with the Street View House Numbers (SVHN) Dataset: Figure 12 depicts the time taken for each epoch during training for no offload and proposed designs. **O-DA**, **O-MV**, and **O-Hy** result in 7%, 5.5%, and 10.1% speedups, respectively.

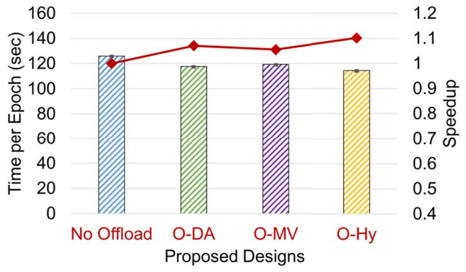


Fig. 12. Performance characterization of training ResNet-56 model with SVHN dataset using proposed designs

Training ShuffleNet with the Tiny ImageNet Dataset: Figure 13 depicts the time taken for each epoch during training for no offload and proposed designs. **O-DA**, **O-MV**, and **O-Hy** result in 12.5%, 1.2%, and 8.9% speedups, respectively.

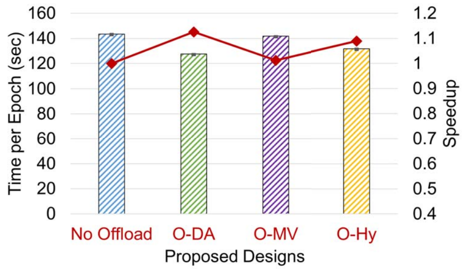


Fig. 13. Performance characterization of training ShuffleNet model with Tiny ImageNet dataset using proposed designs

Experiments conducted in this sub-section demonstrate that **O-DA** achieves better performance when data preprocessing is significant to provide overlap to DNN training executing on the host processor. The other two designs namely **O-MV** and **O-Hy** achieve good performance if DNN training time—that includes forward and backward pass—dominate the total execution time. In this case, the data preprocessing time is typically negligible.

C. Multiple Node Experiments

This sub-section presents performance evaluation of our proposed designs using various DL models and datasets on multiple nodes of the HPCAC system. The idea here is to understand the scaling behavior of our proposed designs. We make use of the lessons learned from single node comparisons done as part of Section VI-B and appropriately choose best performing design for particular DL model and dataset. We learned from Section VI-B that **O-MV** and **O-Hy** generate best performance when data preprocessing is negligible as compared to DNN training (forward and backward pass stages). Otherwise, **O-DA** performs the best.

Training ResNet-20 with the CIFAR-10 Dataset: Figure 14 depicts the performance of the **O-DA** design up to 16 nodes of the HPCAC system. The number of MPI processes on each node remains the same—8 MPI processes (each with 4 OpenMP threads) on the host and 8 MPI processor on the DPU—as described in Section VI-B. Results indicate that the **O-DA** design achieves an average speedup of 13.9% on $1-16$ nodes. The maximum speedup of 15% was achieved on 4 nodes.

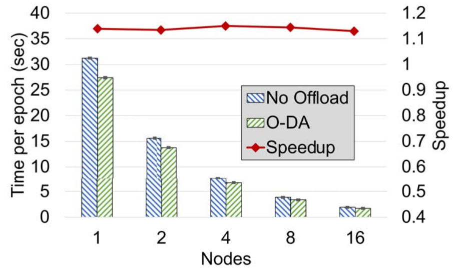


Fig. 14. Scalability evaluation of ResNet-20 on increasing number of nodes on the HPCAC system for the CIFAR-10 dataset using O-DA

Training ResNet-56 with the SVHN Dataset: Figure 15 shows the performance of the **O-Hy** design up to 16 nodes of the HPCAC system. Results indicate that the **O-Hy** design achieves an average speedup of 9.7% on $1-16$ nodes. The maximum speedup of 11.2% was achieved on 2 nodes. The speedup achieved on 16 nodes is 9.3%.

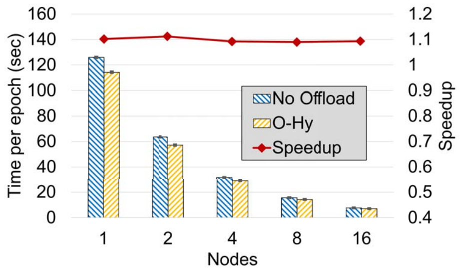


Fig. 15. Scalability evaluation of ResNet-56 on increasing number of nodes on the HPCAC system for the SVNH dataset using O-Hy

Training ShuffleNet with the Tiny ImageNet Dataset: Figure 16 shows the performance of the **O-DA** design up to 16 nodes of the HPCAC system. Results indicate that the **O-DA** design achieves an average speedup of 11.1% on $1-16$ nodes. The speedup achieved on 16 nodes is 10.2%.

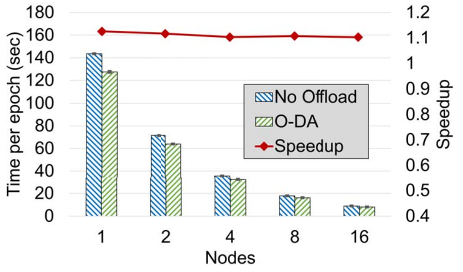


Fig. 16. Scalability evaluation of ShuffleNet on increasing number of nodes on the HPCAC system for the Tiny ImageNet dataset using O-DA

VII. Related Work

Several studies [20]–[23] have evaluated GPU-based DNN training as GPU clusters are mostly used for training DNN.

However, CPUs can also be used for DNN training as Intel has optimized its architecture using the MKL-DNN library. Few studies [3], [7], [8], [24], [25] exist in the literature that evaluates CPU-based DNN training. In this paper, we evaluate CPU-based DNN training and exploit BlueField-2 DPUs to accelerate the DNN training pipeline. Data preprocessing backends like PyTorch's Dataloader and TFRecord [26] are implemented by DL frameworks to facilitate DL applications. Several solutions [27]–[30] have been proposed data preprocessing for GPU-based DNN training using CPUs and FPGAs. However, BlueField-2 DPUs present different challenges as they lack support for unified and shared memory. In this paper, we explore different offloading designs for DPUs in CPU-based DNN training and offload model validation in addition to data preprocessing.

VIII. Conclusion

In this paper, we characterized and explored how one can take advantage of the additional ARM cores on the Bluefield DPUs to intelligently accelerate different phases of DL training. We proposed four designs: 1) Offload Naive, 2) Offload Data Augmentation, 3) Offload Model Validation, and 4) Offload Hybrid to offload different phases of DL training to the Bluefield DPUs. The reported max speedup improvements are 15%, 12.5%, and 11.2% for training the ResNet-20 model on the CIFAR-10 dataset, ShuffleNet model on the Tiny Imagnet dataset, and ResNet-56 model on the SVHN dataset respectively. Our experimental results show that the proposed designs are able to deliver up to 15% improvement in DL training time. To the best of our knowledge, this is the first work to explore the use of DPUs to accelerate DL training. In the future, we would like to study the offloading schemes for Machine Learning and Data Analytics algorithms and use DPUs to improve the performance.

References

[1] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. Hashmi, and D. Panda, "Bluesmpi: Efficient mpi non-blocking alltoall offloading designs on modern bluefield smart nics," June 2021.

[2] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," *CoRR*, vol. abs/1802.01548, 2018.

[3] A. A. Awan, A. Jain, Q. Anthony, H. Subramoni, and D. K. Panda, "Hypar-flow: Exploiting mpi and keras for scalable hybrid-parallel dnn training using tensorflow," 2019.

[4] A. Jain, A. Awan, A. Aljuhani, J. Hashmi, Q. Anthony, H. Subramoni, D. Panda, R. Machiraju, and A. Parwani, "Gems: Gpu-enabled memory-aware model-parallelism system for distributed dnn training," in *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 621–635, IEEE Computer Society, 2020.

[5] A. Jain, T. Moon, T. Benson, H. Subramoni, S. Jacobs, D. Panda, and B. Essen, "Super: Sub-graph parallelism for transformers," in *35th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2021.

[6] "Mellanox BlueField." https://docs.mellanox.com/x/iQO3. [Online; accessed July 21, 2021].

[7] A. Jain, A. Awan, Q. Anthony, H. Subramoni, and D. Panda, "Performance characterization of dnn training using tensorflow and pytorch on modern clusters," September 2019.

[8] A. Jain, A. A. Awan, H. Subramoni, and D. K. Panda, "Scaling tensorflow, pytorch, and mxnet using mvapich2 for high-performance deep learning on frontera," in *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, pp. 76–83, 2019.

[9] "High-Performance Center Overview,"

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[11] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *CoRR*, vol. abs/1707.01083, 2017.

[12] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[13] G. H. Alex Krizhevsky, Vinod Nair, "The CIFAR-10 Dataset." Accessed: July 21, 2021.

[14] "The Street View House Numbers (SVNH) Dataset." Accessed: July 21, 2021.

[15] J. J. Fei-Fei Li, Andrej Karpathy, "The Tiny ImageNet Dataset." Accessed: July 21, 2021.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[17] A. Sergeev and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018.

[18] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011. New Computational Methods and Software Tools.

[19] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE. https://mvapich.cse.ohio-state.edu/, 2001. [Online; accessed July 21, 2021].

[20] J. Han, L. Xu, M. M. Rafique, A. R. Butt, and S.-H. Lim, "A quantitative study of deep learning training on heterogeneous supercomputers," September 2019.

[21] A. A. Awan, J. Bedorf, C.-H. Chu, H. Subramoni, and D. K. Panda, "Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation," *arXiv preprint arXiv:1810.11112*, 2018.

[22] A. A. Awan, H. Subramoni, and D. K. Panda, "An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures," in *Proceedings of the Machine Learning on HPC Environments*, MLHPC'17, (New York, NY, USA), pp. 8:1–8:8, ACM, 2017.

[23] P. Kousha, B. Ramesh, K. Kandadi Suresh, C.-H. Chu, A. Jain, N. Sarkauskas, H. Subramoni, and D. K. Panda, "Designing a profiling and visualization tool for scalable and in-depth analysis of high-performance gpu clusters," in *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 93–102, 2019.

[24] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pp. 99–104, Nov 2016.

[25] A. Viebke and S. Pllana, "The potential of the intel (r) xeon phi for supervised deep learning," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 758–765, Aug 2015.

[26] TensorFlow, "Introduction to tfrecords." https://www.tensorflow.org/tutorials/load_data/tfrecord, 2021. [Online; accessed July 21, 2021].

[27] NVIDIA, "Data loading library (dali)." https://developer.nvidia.com/dali, 2021. [Online; accessed July 21, 2021].

[28] NVIDIA, "nvjpeg." https://developer.nvidia.com/nvjpeg, 2021. [Online; accessed July 21, 2021].

[29] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *CoRR*, vol. abs/1706.02677, 2017.

[30] Y. Cheng, D. Li, Z. Guo, B. Jiang, J. Geng, W. Bai, J. Wu, and Y. Xiong, "Accelerating end-to-end deep learning workflow with codesign of data preprocessing and scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1802–1814, 2021.