Optimizing Distributed DNN Training Using CPUs and BlueField-2 DPUs

Arpan Jain, Nawras Alnaasan, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda, *The Ohio State University, Columbus, OH, 43210-1277, USA*

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 central processing units or graphics processing units in a serial fashion due to lack of additional computing resources to offload independent phases of DL training. Recently, Mellanox/NVIDIA introduced the BlueField-2 data processing units (DPUs), which combine the advanced capabilities of traditional application-specific-integrated-circuit-based network adapters with an array of ARM processors. In this article, we explore how to take advantage of the additional ARM cores on the BlueField-2 DPUs. We propose and evaluate multiple novel designs to efficiently offload the phases of DL training to the DPUs. Our experimental results show that the proposed designs are able to deliver up to 17.5% 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.

The hyperscale data centers have been using smart network interface cards (NICs) to offload a variety of functions from the host processor. These functions typically include data and control plane switching, network function virtualization, 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 high performance computing (HPC) community to enable offloading high-value communication and compute operations. Unlike the data center community, the exploration of these DPUs as additional processing elements—along with central processing units (CPUs) and graphics processing units (GPUs)—is still in its infancy. One approach[1] is to offload message passing interface (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. Table 1 depicts relative speedups achieved using CPUs and GPUs—with K80, P100, and V100 GPUs—as compared to using CPUs alone. In the context of this article, we aim to optimize CPU performance of deep learning (DL) workloads using DPUs as additional processing elements.

Distributed DL has become the default approach to achieve models with high accuracy in areas like natural language processing, computer vision, and recommendation systems. Distributed deep neural network (DNN) training can be categorized into three approaches: 1) data parallelism[2]; 2) model parallelism[3]; and 3) hybrid parallelism.[4] Data parallelism creates a model replica on each processing element and conducts forward and backward passes simultaneously. At the end of the backward pass, the model is synchronized using an allreduce operation.

DISTRIBUTED DL HAS BECOME THE DEFAULT APPROACH TO ACHIEVE MODELS WITH HIGH ACCURACY IN AREAS LIKE NATURAL LANGUAGE PROCESSING, COMPUTER VISION, AND RECOMMENDATION SYSTEMS.

DL uses DNNs to learn the relationship between input and output by training it on a large corpus of

0272-1732 © 2021 IEEE
Digital Object Identifier 10.1109/MM.2021.3139027
Date of publication 29 December 2021; date of current version 28 March 2022.

TABLE 1. Speedup on CIFAR-10 data set relative to CPU.

Model	NVIDIA K80	NVIDIA P100	NVIDIA V100
ResNet-20	1.12X	3.8X	5.8X
ResNet-56	0.55X	1.9X	3.3X

data.[5] 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 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 the Math Kernel Library-Deep Neural Network (MKL-DNN). Hence, CPU-based DNN training is gaining a lot of traction in the community. In this article, we explore the possibility of offloading different phases of DL training to DPUs for data parallelism.

CONTRIBUTIONS

In this article, we optimize various parameters such as *OMP_NUM_THREADS*, the number of processes per node, and the batch size for the PyTorch DL framework to get good performance on CPU and DPU. We use multiple process multiple data 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. To the best of our knowledge, this is the first work to explore the use of DPUs to accelerate DL training on a multinode heterogeneous CPU+DPU cluster.

To summarize, this article provides the following contributions:

- Proposing multiple novel designs for offloading different phases of DL training to the BlueField-2 DPUs.
- Performance evaluations, with weak and strong scalability analysis, of the proposed designs with multiple convolution neural network (CNN) and transformer models (ResNet-20, ResNet-56, ShuffleNet, and DistilBERT) with four data sets.
- Obtained speedup improvements up to 15%, 12.5%, 11.2%, and 8.6% for training the ResNet-20 model on the CIFAR-10 data set, ShuffleNet model on the Tiny ImageNet data set, ResNet-56 model on the SVHN data set, and DistilBERT for sentiment analysis, respectively.
- Scaled proposed designs to 16 nodes, studied the impact of underlying filesystem, and achieved consistent improvement up to 17.5% for multi-node experiments.

PROPOSED ADVANCE OFFLOADING DESIGNS

DL training consists of several phases and steps. Each training 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 propose advanced offloading designs. The basic offloading design that offloads the entire DNN training pipeline to DPUs has been discussed in the earlier version of this article[6] and has been omitted here.

Design 1: Offload Data Augmentation (O-DA)

We offload the reading of training data from memory and data augmentation on input data to DPUs (see Figure 1). There are two types of processes: 1) training process and 2) data augmentation process. The training process on CPU does forward/backward pass, gradient synchronization (for data parallelism), weight update, and model validation steps. The data augmentation process on DPUs fetch the training data from storage, applies user-defined data augmentation functions, and sends the batch of input and output to the training process on CPU. Since DPUs have eight cores, we limit the number of processes per CPU/DPU to eight.

Training process: We create two buffers of the same size as input and output to overlap commutation overhead with forward and backward passes 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 passes.

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.

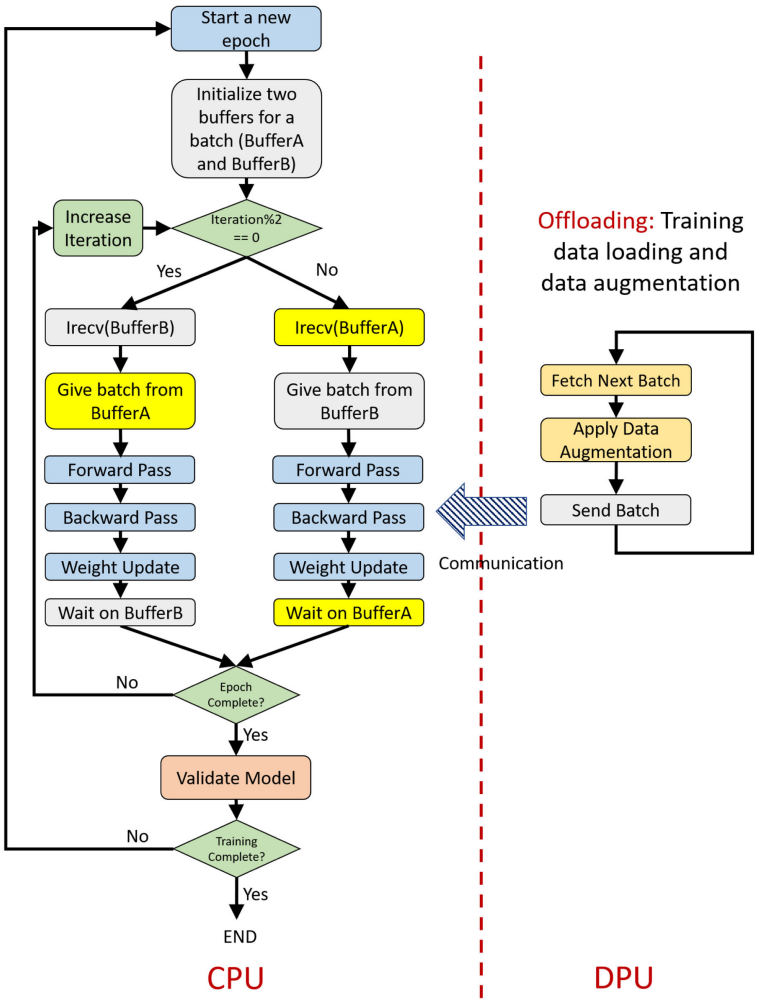

FIGURE 1. Flowchart of the proposed offload data augmentation (O-DA) design. It offloads the reading of data from memory and data augmentation functions to DPU.

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 the 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 2 shows the offloading of model validation to DPU for one CPU and DPU process. We create two types of processes in this design: 1) the training process on CPU and 2) the testing process on DPU.

Training process: In this strategy, the training process fetches the training data from memory, applies data augmentation, and performs forward/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 forward/backward pass on CPUs. This leads to degradation as time per epoch is the maximum of the model validation on DPU and the forward/backward pass 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

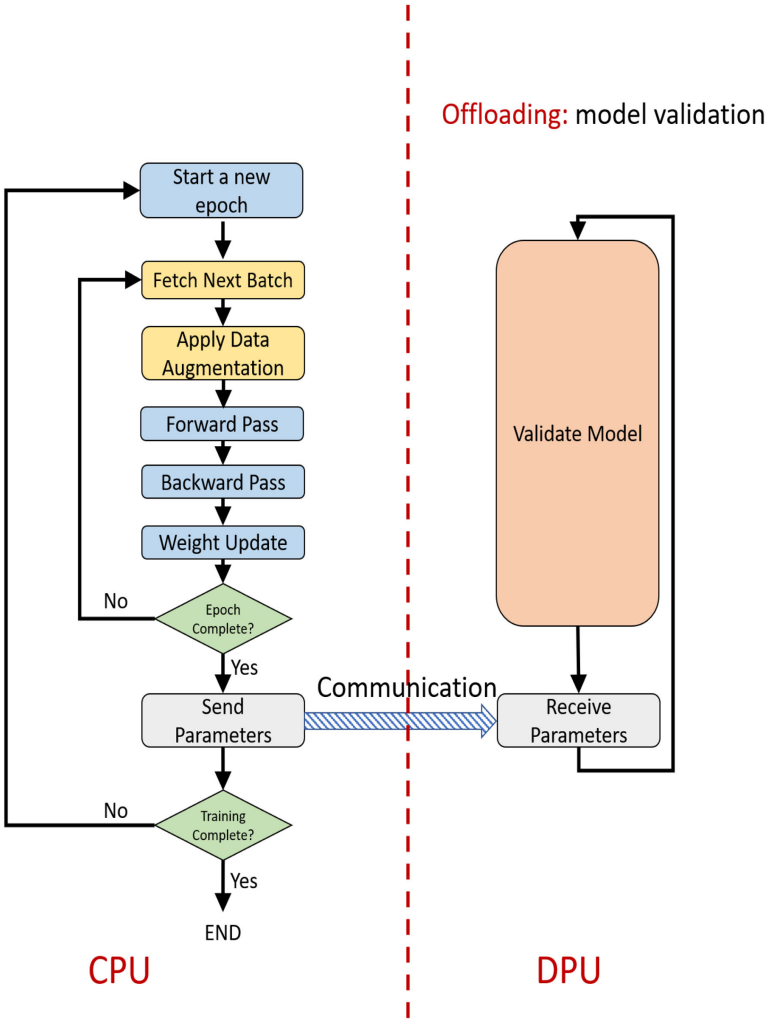

FIGURE 2. Flowchart of the proposed offload model validation (O-MV) design. It offloads the validation of the model on the testing data set and overlaps the computation with the training of the next epoch.

good overlap. We use an analytical model to distribute validation data between CPU and DPU. First, we calculate time per batch for training on CPU and time per batch for validation on CPU and DPU; then, we estimate the total training time for an epoch on CPU and divide it with the time per batch for testing on DPUs. Further, we found that Horovod initializes a background thread to progress the communication. Testing processes do not use Horovod; therefore, we initialize the Horovod on a subset of all the processes (training processes only). This small optimization improved the time per batch for validation on DPUs by 30%–40%.

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 passes (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 3 shows the flow diagram of hybrid offloading 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 the 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 processes on each DPU that fetch training data from memory, augment the data, and send it to

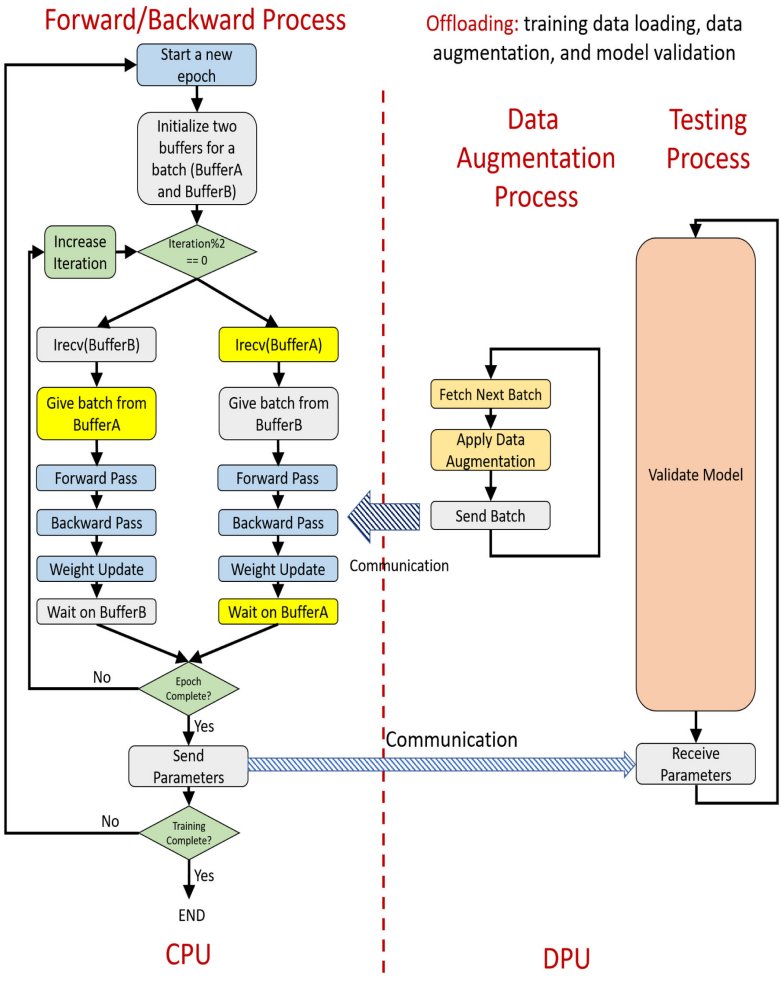

FIGURE 3. Offload hybrid (O-Hy) (data augmentation and model validation) to DPU.

TABLE 2. Speedup of the proposed offloading designs over CPU on a single node.

Model (data set)	O-DA	O-MV	O-Hy
ResNet-20 (CIFAR-10)	13.8%	3.1%	0%
ResNet-56 (SVHN)	7%	5.5%	10.1%
ShuffleNet (Tiny-ImageNet)	12.5%	1.2%	8.9%

the forward/backward process on CPU. 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 data set. They overlap the validation with the training of the next epoch. We divide testing data among testing processes and use the same optimizations as discussed in the "Design 2: Offload Model Validation (O-MV)" section.

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 data sets. These proposed approaches include Design #1: O-DA, Design #2: O-MV, and Design #3: O-Hy. We also add No Offload to our evaluation in order to provide baseline performance in the absence of DPUs.

Experimental Setup

We used the HPC Advisory Council High-Performance Center (HPCAC) cluster for our evaluation. HPCAC has 32 nodes with BlueField-2 network adapters

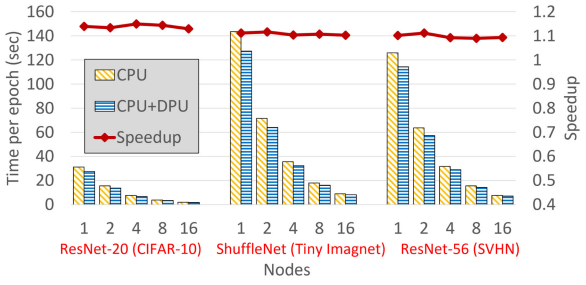

FIGURE 4. Multinode performance characterization of CNNs on different data sets using weak scaling.

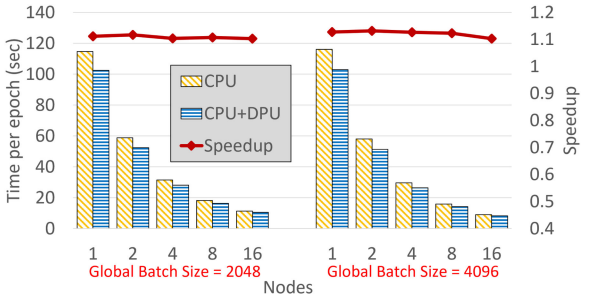

FIGURE 5. Multinode performance characterization of CNNs on different data sets using strong scaling.

equipped with eight ARM cores. Each BlueField-2 adapter is equipped with Mellanox MT41682 EDR ConnectX-5 HCAs (100-Gb/s data rate) with PCI-Ex Gen3 interfaces. The host is equipped with the Broadwell series of Xeon dual-socket, 16-core processors operating at 2.60 GHz with 128-GB RAM.

DNN: ResNet,[7] ShuffleNet, and DistilBERT.
Data set: CIFAR-10, Street View House Numbers (SVHN) Data set, and Tiny ImageNet.
Software libraries: PyTorch v1.9, Horovod v0.21,[8] HuggingFace v4.9.1, TorchVision v0.11, MPI4py v3.0.3, and MVAPICH2 2.3.6 MPI library

CNN Experiments

This section presents performance evaluation of our proposed designs using various CNNs and data sets on a single and multiple nodes. The idea here is to understand the performance on a single node and the scaling behavior of our proposed design on various data sets. We launched eight MPI processes on the host processor and eight additional MPI processes on the BlueField-2 DPU to fully exploit the available eight ARM cores. These eight MPI processes are used for different purposes in our proposed designs.

Single node: Table 2 show the performance of proposed designs for various CNNs. Experiments 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 passes—dominate the total execution time.

Weak scaling: Figure 4 shows the weak scaling experiments for CNNs used in single node experiments. We make use of the lessons learned from single node comparisons and appropriately choose the best-performing design for a particular DL model and data set. We report a consistent speedup for weak scaling experiments. We achieved the maximum speedup of 15% on four nodes for ResNet-20 on the CIFAR-10 data set for four nodes.

Strong scaling: In weak scaling, the batch size per process remains, while in strong scaling, the global batch size remains the same. Weak scaling is good for scalability, while strong scaling is good for achieving better training/validation loss. Therefore, we present results for both weak and strong scaling. Figure 5 shows the speedup for strong scaling with 2048 and 4096 batch sizes on the Tiny ImageNet data set.

Transformer Model Experiments

Transformer represents a different spectrum of state-of-the-art models for sequential data and tasks like natural language processing and speech recognition. Our experiments found that real-time data augmentation is negligible in transformers and model validation takes a significant amount of time. Therefore, we used the proposed O-MV design to accelerate the training. We evaluated DistilBERT on a synthetic data set similar to the IMDB movie review data set. NLP has several large data sets with the number of samples ranging from 10,000 to 1,000,000. In our experiments, we limit the number of samples to 100,000 as the size of the data set does not affect the speedup in O-MV (training and validation data split ratio remains the same).

Multinode experiments: We provide weak and strong scaling performance numbers for the DistilBERT on 1–16 nodes. Our evaluations found that four PPN and $OMP\_NUM\_THREADS = 2$ give the best time per batch for model validation on DPUs. We split validation between CPU and DPU as validating the model on DPU only takes more time than training on CPUs. Figure 6 shows weak and strong scaling speedup for the DistilBERT transformer model. We achieve up to 8.6% improvement using the proposed O-MV design.

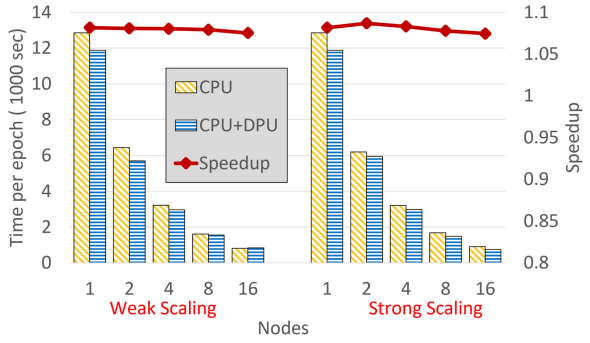

FIGURE 6. Multinode performance characterization of DistilBERT.

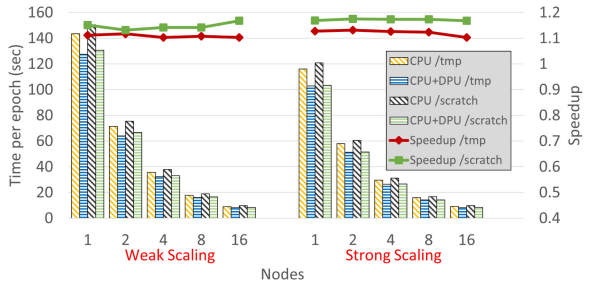

FIGURE 7. Effect of filesystem on data-loading and overall training time.

Effect of File System on Data-Loading

We evaluate the effect of the filesystem (`tmp` and `scratch`) on the overall DNN training. `tmp` is a temporary small filesystem exclusive to a computing node, while `scratch` is a parallel filesystem accessible from all computing nodes. Thus, smaller data sets can be copied to `tmp` to get better performance, while larger data sets can only be placed on `scratch`. Figure 7 shows the performance benefits of the proposed O-DA design when the Tiny ImageNet data set is kept on two different filesystems. `scratch` filesystem increases the data reading time, which increases the overall training time. However, O-DA offloads data reading to DPUs, thereby negating the effect of increased data reading time. We show up to 17.5% improvement for ShuffleNet training when Tiny ImageNet data set is stored in the `scratch` filesystem.

Discussion

While data loading and augmentation overhead depends on the size of data samples, model validation is also affected by the DNN as it involves forward pass. However, if DNN is very large, then data loading becomes insignificant. In addition, batch size is critical for overall training time; nonetheless, its impact is negligible on data loading as the same number of data samples is loaded in each epoch. Since training is performed on CPUs, batch size can be increased for very-large DNNs due to sufficient host memory. The proposed O-DA design improves training time when DNN is small. O-Hy gives better performance for large DNN as data loading overhead decreases. O-MV design should be used when the data set does not need augmentation and can be loaded directly to memory.

RELATED WORK

Several studies[2,9] have evaluated CPU and GPU-based DNN training. Data preprocessing backends like PyTorch's Dataloader and TFRecord are implemented by DL frameworks to facilitate DL applications. Several solutions[5,10] have been proposed to perform 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 article, we explore different offloading designs for DPUs in CPU-based DNN training and O-MV in addition to data preprocessing.

CONCLUSION

In this article, we characterized and explored how one can take advantage of the additional ARM cores on the Bluefield-2 DPUs to intelligently accelerate different phases of DL training. We proposed three designs: 1) O-DA; 2) O-MV; and 3) O-Hy to offload different phases of DL training to the Bluefield DPUs. The reported max speedup improvements are 15%, 12.5%, 11.2%, and 8.5% for training the ResNet-20 model on the CIFAR-10 data set, ShuffleNet model on the Tiny Imagnet data set, ResNet-56 model on the SVHN data set, and DistilBERT for sentiment analysis, respectively. We showed consistent improvement for CNNs and transformer models with weak and strong scaling on multiple nodes. Further, we explored the effect of the filesystem on overall training time and reported up to 17.5% improvement with proposed designs. To the best of our knowledge, this is the first work to explore the use of DPUs to accelerate DL training.

IN THIS ARTICLE, WE CHARACTERIZED AND EXPLORED HOW ONE CAN TAKE ADVANTAGE OF THE ADDITIONAL ARM CORES ON THE Bluefield-2 DPUs TO INTELLIGENTLY ACCELERATE DIFFERENT PHASES OF DL TRAINING.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grants #1818253, #1854828, #1931537, #2007991, and #2018627; and in part by the XSEDE Resource Allocation Committee under Grant #NCR-130002.

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," in *Proc. Int. Conf. High Perform. Comput.*, Jun. 2021, pp. 18–37.
2. A. Jain, A. Awan, Q. Anthony, H. Subramoni, and D. Panda, "Performance characterization of DNN training using tensorflow and PyTorch on modern clusters," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2019, pp. 1–11.
3. A. Jain *et al.*, "GEMS: GPU-enabled memory-aware model-parallelism system for distributed DNN training," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 621–635.
4. A. Jain *et al.*, "Super: Sub-graph parallelism for transformers," in *Proc. 35th IEEE Int. Parallel Distrib. Process. Symp.*, May 2021, pp. 629–638.
5. P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv:1706.02677*.
6. A. Jain, N. Alnaasan, A. Shafi, H. Subramoni, and D. Panda, "Accelerating CPU-based distributed DNN training on modern HPC clusters using bluefield-2 DPUs," in *Proc. IEEE Symp. High-Perform. Interconnects*, 2021, pp. 17–24.
7. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
8. A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," 2018, *arXiv:1802.05799*.
9. J. Han, L. Xu, M. M. Rafique, A. R. Butt, and S.-H. Lim, "A quantitative study of deep learning training on heterogeneous supercomputers," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2019, pp. 1–12.
10. Y. Cheng *et al.*, "Accelerating end-to-end deep learning workflow with codesign of data preprocessing and scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1802–1814, Jul. 2021.

ARPAN JAIN is a Graduate Research Assistant with the Network-Based Computing Laboratory, Columbus, OH, USA. His current research interests include intersection of deep learning and high-performance computing. Jain received an M.S. degree in information and communication technology from ABV Indian Institute of Information Technology and Management, Gwalior, India. He is currently working toward a Ph.D. degree with The Ohio State University, Columbus, OH, USA, working with Prof. D. K. Panda. He is a Graduate Student Member of IEEE. Contact him at jain.575@osu.edu.

NAWRAS ALNAASAN is currently working toward a Ph.D. degree in computer science and engineering with The Ohio State University, Columbus, OH, USA. He is also exploring creative approaches to accelerate distributed DNN training. His research focuses on high-performance deep learning. Alnaasan received a B.S. degree in computer science and engineering from the Ohio State University. Contact him at alnaasan.1@osu.edu.

AAMIR SHAFI is a Research Scientist with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA, where he is involved in the High Performance Big Data project. He was a Fulbright visiting scholar with MIT, Cambridge, MA, USA, where he worked on the award-winning Cilk technology. He has co-designed and co-developed a Java-based MPI-like library called MPJ Express. His current research interests include architecting robust libraries and tools for big data computation with emphasis on machine and deep-learning applications. Shafi received a Ph.D. degree in computer science from the University of Portsmouth, Portsmouth, U.K., in 2006. Contact him at shafi.16@osu.edu.

HARI SUBRAMONI has been a Research Scientist with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA, since September 2015. He is conducting research and working on the design and development of MVAPICH2, MVAPICH2-GDR, and MVAPICH2-X software packages. Subramoni received a Ph.D. degree in computer science from The Ohio State University, in 2013. He is a Member of IEEE. Contact him at subramon@cse.ohio-state.edu.

DHABALESWAR K. PANDA is a Professor of computer science and engineering with The Ohio State University, Columbus, OH, USA. His research interests include parallel computer architecture, high-performance networking, InfiniBand, exascale computing, Big Data, programming models, GPUs and accelerators, high-performance file systems and storage, virtualization, deep learning, and cloud computing. He is an IEEE Fellow and a Member of ACM. Contact him at panda@cse.ohio-state.edu.