Accelerate Scientific Deep Learning Models on Heterogeneous Computing Platform with FPGA

Chao Jiang[1,*], *David* Ojika[1,5,**], *Sofia* Vallecorsa[2,***], *Thorsten* Kurth[3], Prabhat[4,****], *Bhavesh* Patel[5,†], and *Herman* Lam[1,‡]

[1]SHREC: NSF Center for Space, High-Performance, and Resilient Computing, University of Florida
[2]CERN openlab
[3]NVIDIA
[4]National Energy Research Scientific Computing Center
[5]Dell EMC

Abstract. AI and deep learning are experiencing explosive growth in almost every domain involving analysis of big data. Deep learning using Deep Neural Networks (DNNs) has shown great promise for such scientific data analysis applications. However, traditional CPU-based sequential computing without special instructions can no longer meet the requirements of mission-critical applications, which are compute-intensive and require low latency and high throughput. Heterogeneous computing (HGC), with CPUs integrated with GPUs, FPGAs, and other science-targeted accelerators, offers unique capabilities to accelerate DNNs. Collaborating researchers at SHREC[1]at the University of Florida, CERN Openlab, NERSC[2]at Lawrence Berkeley National Lab, Dell EMC, and Intel are studying the application of heterogeneous computing (HGC) to scientific problems using DNN models. This paper focuses on the use of FPGAs to accelerate the inferencing stage of the HGC workflow. We present case studies and results in inferencing state-of-the-art DNN models for scientific data analysis, using Intel distribution of OpenVINO, running on an Intel Programmable Acceleration Card (PAC) equipped with an Arria 10 GX FPGA. Using the Intel Deep Learning Acceleration (DLA) development suite to optimize existing FPGA primitives and develop new ones, we were able accelerate the scientific DNN models under study with a speedup from 2.46x to 9.59x for a single Arria 10 FPGA against a single core (single thread) of a server-class Skylake CPU.

1 Introduction

The concept diagram for the HGC workflow for deep learning is shown in Figure 1. The HGC workflow consists of three stages: Data Analysis and Pre-processing, Model Training, Deployment and Inferencing. The Data Analysis and Pre-processing stage converts raw data from an application of interest into a form that is suitable for model training using

*e-mail: jc19chaoj@ufl.edu
**e-mail: davido@ufl.edu
***e-mail: sofia.vallecorsa@cern.ch
****e-mail: prabhat@lbl.gov
†e-mail: bhavesh.a.patel@dell.com
‡e-mail: hlam@ufl.edu

existing training frameworks. Current pre-processing methods include data cleaning, data normalization, and data augmentation[1]. In the HGC workflow, the pre-processed data is used as inputs to the training tools in the Model Training stage. Currently, popular open-source frameworks for training these models include TensorFlow[2], Keras[3], Caffe[4], and BigDL[5]. The output of the Training stage is trained models which are used in inference engines in the Deployment and Inferencing stage. Some of the common approaches for running inferencing primarily use CPUs and GPUs; but recently there has been much interest in using FPGAs (field programmable gate arrays) for inferencing.

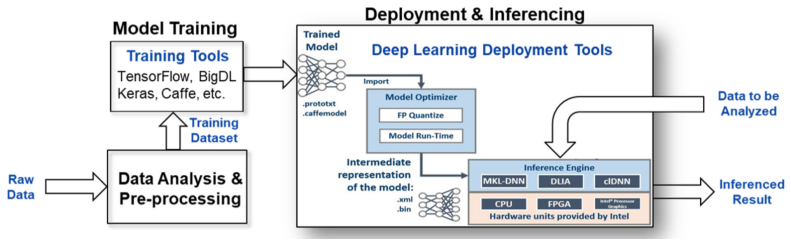

Figure 1: Heterogeneous computing workflow for machine learning

This paper will focus on how we made use of Intel Arria 10 FPGAs for inferencing and what is the workflow behind it. After a brief survey of recent and state-of-the-art FPGA deep-learning acceleration tools available in research and commercially in Section 2, we will describe our experimental setup in Section 3. The hardware platform for the inference engine is the Intel Programmable Acceleration Card (PAC), equipped with an Arria 10 GX FPGA. The PAC card is installed in a Dell server equipped with an Intel gold-level Skylake CPU. Intel distribution of OpenVINO toolkit is used to optimize and deploy the trained models onto the FPGA. Also in Section 3, we will describe the three state-of-the-art, case-study DNN models used for this study: HEP-CNN, CosmoGAN, and 3DGAN. HEP-CNN[6, 7] is a deep-learning model used by NERSC to identify new particles produced during collision events at particle accelerators such as the Large Hadron Collider (LHC). CosmoGAN[8], also under study by NERSC, is used to generate cosmology weak lensing convergence maps to study GAN model for science applications. The 3DGAN model [9] is a generative model under study by CERN openlab to replace the Monte Carlo method for particle collision simulations.

In Section 4, we present the experimental setup, including Intel's OpenVino Toolkit, Deep Learning Accelerator (DLA), and the hardware platform. Also we present and discuss the initial findings using the native DLA runtime. The preliminary results from the studies described in this section and a demonstration of the HGC workflow were submitted to and declared the winner of the first-ever Dell EMC AI Challenge in 2018[10].

In Section 5, we described how we used the DLA development suite to *optimize existing* FPGA primitives in OpenVINO to improve performance and *develop new* primitives to enable new capabilities for FPGA inferencing. For the scientific DNN models under study, we were able to demonstrate a speedup from 2.46x to 9.59x for a single Arria 10 FPGA against a single core (single thread) of a server-class Skylake CPU. Finally, Section 6 will provide the conclusions of the paper and a discussion of future work.

2 Related Works

Although CPUs and GPUs have been widely used for DNN inferencing, inference engines accelerated with FPGAs have recently emerged. Recent improvements in FPGA technologies

greatly increased the performance for DNN applications, e.g., with a reported performance of 9.2 TFLOPS for Intel Stratix 10 FPGA[11]. Furthermore, FPGAs have other advantages important to many mission-critical applications such as low latency and energy efficiency. As a result, the amount of research and development on deploying and accelerating DNN models on FPGAs in recent years has grown, demonstrating great interest in both academia and industry.

Notable tools developed in the research community include PipeCNN[12] and hls4ml [13]. PipeCNN is an OpenCL-based FPGA accelerator designed for large-scale convolutional neural networks (CNNs). The main goal of PipeCNN is to provide an FPGA accelerator architecture of deeply pipelined CNN kernels to achieve improved throughput in the inference stage. Unlike previous OpenCL design, memory bandwidth is minimized by pipelining CNN kernels. Efficiency is enhanced by using task-mapping techniques and data reuse. Developed at Fermilab, hls4ml[13] is a deep neural network compiler based on HLS (High-level Synthesis language). The input to hls4ml is a fully connected neural network trained from conventional training frameworks such as Keras and PyTorch. The network is translated to Vivado HLS and then compiled for the target FPGA. For the initial result in using this framework, the researchers focused on using FPGA for machine learning in an application of real-time event reconstruction and filtering in the Large Hadron Collider at CERN.

The success of deploying and accelerating DNN models on FPGAs resulted in commercial offerings of these tools from both major FPGA vendors. OpenVINO, from Intel/Altera[14], is a comprehensive toolkit designed to support deep learning, computer vision, and hardware acceleration using heterogeneous (CPU, GPU, FPGA) platforms. The OpenVINO toolkit is used extensive in this project and will be described in detail in Section 3. With the recent acquisition of DeePhi, Xilinx provides the Deep Neural Network Development Kit (DNNDK)[15] to enable the acceleration of the deep learning algorithms in FPGAs and SoCs. At the heart of the DNNDK is the deep learning processor unit (DPU). The DNNDK deep learning SDK is designed as an integrated framework which aims to simplify and accelerate deep learning applications development and deployment for Xilinx DPU platforms.The basic stages of deploying a deep learning application into a DPU are: compress the DNN model to reduce the model size without loss of accuracy; compile the DNN model into DPU instruction code; create an application using DNNDK (C/C++) APIs; use the hybrid compiler to compile and deploy the hybrid DPU application on the target DPU platform.

3 Overview of Case Study

As illustrated in Figure 1, the heterogeneous computing workflow for DNN consists of three stages: 1) Data Analysis and Pre-processing, 2) Model Training, and 3) Deployment and Inferencing. In this section, we introduced two case-study models that can be partially inferenced with the "native" (original Intel distribution) OpenVINO: HEP-CNN and CosmoGAN, and one case-study model that cannot: 3DGAN. Also in this section, we described how they were pre-processed and trained in preparation for the Deployment and Inferencing stage. The initial inferencing results using the native OpenVINO will be presented in section 4.

3.1 HEP-CNN

HEP-CNN[6, 7] was developed as a proof-of-concept study for improved event selection at particle collider experiments. These experiments generate large amount of data of a detector snapshot after a number of particle collisions. Most of the events can be explained by the well understood Standard Model of Particle Physics, also referred to as *background*. The challenge is to find and select events which potentially contain candidates for new physics.

The training data was obtained by coupling the Pythia[16] event generator to the Delphes[17] fast detector simulator. The cylindrical data is represented as a 2D image of size 224x224, where the two dimensions represent the binned azimuth angle and pseudorapidity[18] coordinates. The three input channels are given by the hadron and electromagnetic calorimeter energy deposits as well as the multiplicity of reconstructed tracks from the pixel detector.

HEP-CNN is comprised of 5 convolution and max-pooling layers with Leaky ReLU activations[19, 20]. The kernel employs 128 filters per layer. The final set of layers consists of an average pooling across the dimensions output image followed by a fully connected layer with softmax activation which performs the binary classification. The model outperforms its benchmark, i.e. a hand-crafted decision tree, by more than 2x in true positive rate at the same false negative rate.

3.2 CosmoGAN

CosmoGAN[8] is a deep convolutional generative adversarial network (GAN) which was designed to serve as an inexpensive emulator for cosmological simulations which traditionally consist of three-dimensional n-body simulations followed by ray-tracing steps in order to obtain two-dimensional weak gravitational lensing maps.

It is an unconstrained GAN which is able to reproduce these mass maps to very high statistical accuracy (cf. [8]) for a fixed set of cosmological parameters. The network input is a 64-dimensional vector of uncorrelated gaussian noise, followed by a fully connected layer to cross-correlate all inputs, followed by a series of four transposed convolutions, leading to a single 256x256 output image. Each inner layer is batch-normalized[21] and uses Leaky ReLU activation, while the output layer uses a tanh activation.

3.3 3DGAN

3DGAN is a three-dimensional convolutional generative adversarial networks which represents the first application of such model to the simulation of high granularity electromagnetic calorimeters. The model can be passed as input a particle type, energy and trajectory, and will produce an accurate simulation of the corresponding particle detector output. The training data is based on pseudo-data simulated with GEANT4 [22] in the proposed Linear Collider Detector (LCD) for the CLIC accelerator [23]. The LCD consists of a regular grid of 3D cells with cell sizes of 5.1 mm^3 and an inner calorimeter radius of 1.5 m. Individual electron, photon, charged pion, and neutral pion particles are shot into the calorimeter at various energies and at various angles to the calorimeter surface. For each event we take a $25 \times 25 \times 25$ cell slice of the electromagnetic calorimeter (ECAL) and store them as two 3D arrays containing information about the energy deposited in each cell.

The 3DGAN generator and discriminator models consist of four 3D convolution layers. Leaky ReLU activation functions are used for the discriminator network layers. A batch normalization layer is added after all activations except the first layer. The output of the final convolution layer is flattened and connected to a sigmoid neuron corresponding to real/fake output of GAN as well as a linear unit for energy regression. The network was trained for 30 epochs using the RMSprop. Results show a remarkable agreement to standard Monte Carlo output [9].

4 Experimental Setup

In this section, we described the platform setup for FPGA-based inferencing used in this study: the OpenVINO deployment tool, Deep Learning Accelerator, and hardware platform. The initial experiment results are also shown in this section.

4.1 OpenVINO Toolkit

As shown in Figure 1, OpenVINO consists of two parts: Model Optimizer and Inference Engine. The OpenVINO software is built to emulate the Open Visual inference and neural network optimization. The OpenVINO toolkit extends the workload across Intel hardware and maximizes performance. The Model Optimizer is a cross-platform, command-line tool that facilitates the transition between the training and deployment environment on a target inference engine. The input to the Model Optimizer is a network model trained using one of the supported frameworks. It performs static model analysis and adjusts the input deep learning models for optimal execution on end point target devices, which can be a CPU, GPU, FPGA, or a combination (HETERO). The output of the Model Optimizer is an Intermediate Representation (IR) suitable as input to the selected target Inference Engine. The Inference Engine is a C++ library with a set of C++ classes to infer data (images) to obtain a result. The C++ library provides an API to read the IR, set the input and output formats, and execute the model on devices. In our study, our goal in the Deployment and Inferencing stage is to deploy the trained model on an FPGA to accelerate the classification process.

4.2 Deep Learning Accelerator

In order to customize the FPGA architecture for our needs, we acquired (under NDA) the Intel Deep Learning Accelerator (DLA) developer suite, which is the underlying tool that enables the inferencing of DNN models on FPGAs with OpenVINO. DLA consists of a high-level API (DLIA plugin) that interacts with OpenVINO's inference engine and an FPGA bitstream that creates the architecture shown in Fig. 2.

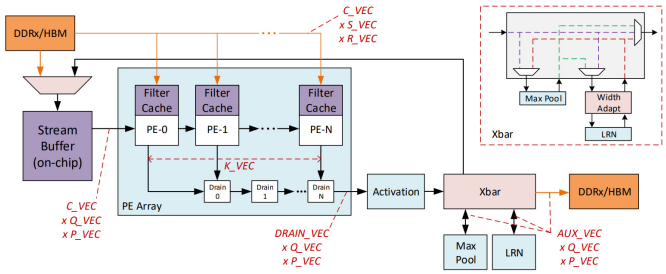

Figure 2: DLA architecture[24]

The architecture contains a stream buffer, a PE (Processing Element) array, and various other modules that compute activation function, max-pooling, and normalization (LRN). The stream buffer takes advantage of the high bandwidth internal RAM of the FPGA, preparing the input data for the PE array. The PE array performs matrix multiplications and accumulations by utilizing DSP resource of the FPGA. DLA inferences a DNN model by first separating it into multiple sub-graphs, which typically consist of a convolutional layer, an activation layer, a max-pooling layer, or a normalization layer. The sub-graphs are then iteratively processed.

The DLA FPGA architecture (Fig. 2) can be customized for inferencing different scientific DNN models in our study. We will see in Section 5 how we implemented new primitives in DLA and how various configurations can affect performance.

4.3 Hardware Platforms

The hardware platform used in this study for the FPGA-accelerated inference engine is the Intel Programmable Acceleration Card (PAC). The PAC card contains an Arria 10 GX, a moderate-sized FPGA fabricated using 20 nm process technolgy. The PAC card is installed in a Dell server equipped with a server-class Intel Gold 6130 Skylake CPU (14 nm process technology), running at a clock speed of 2.1 GHz. The Skylake is a dual-socket CPU, with 16 cores per socket, and 2 threads per core. Performance comparisons to be presented in this paper will be with a single Arria 10 FPGA versus different numbers of Skylake cores and threads.

4.4 Initial Performance

Using native OpenVINO with the DLA (version 2018) implementation, the initial inference performance on HEP-CNN achieved 2.52x speedup against single core/single thread CPU (details in [25]). The speedup would have been greater if not for an unsupported Average Pooling layer of the model. As a result, OpenVINO maps the Average Pooling layer onto the CPU, requiring the FPGA to send the output of the last convolution layer to main memory. The output is then transferred back to FPGA to complete the inference, causing a large data transfer delay.

As for CosmoGAN, the model cannot be fully inferenced on the FPGA due to its unsupported Transposed Convolutional layers. Using CPU as a fallback device to process those layers causes even larger data transfer delay than that of HEP-CNN, resulting in very poor performance (4.8x slower) comparing to single core/single thread Intel SkyLake CPU [25].

5 Customizing DLA

In this section, we described how we customized the DLA development suite (obtained from Intel via NDA) to optimize existing FPGA primitives in OpenVINO and develop new ones to enable new capabilities for FPGA inferencing. The custom DLA was able to improve the performance of HEP-CNN and CosmoGAN. Furthermore, we successfully inferenced 3DGAN model, which could not be inferenced on FPGA using the native DLA.

5.1 Improved Results for HEP-CNN and CosmoGAN

5.1.1 HEP-CNN

For HEP-CNN, we customized the DLA to add an "Average Pooling" primitive to the Pooling module which is attached to the Xbar module of DLA (see Fig. 2) . This customization eliminated the delay from moving data back and forth between the CPU and FPGA and improved the inference performance by 2.6x (vs. native DLA), resulting in a 6.27x speedup against 1 core/1 thread CPU. The details were described in the paper [25].

The inference performance is further improved when using DLA version 2019. This version of DLA replaces the Winograd convolution algorithm with plain convolution which improves FPGA resource and memory bandwidth utilization. We customized DLA 2019 again with the Average Pooling primitive and further improved the inference performance of HEP-CNN by 50%, achieving 9.59x speedup against 1 core/1 thread Intel SkyLake CPU (see Table 1 and Fig. 3).

Table 1: Inference performance of case study models with custom OpenVINO

Models	FPGA*	Speedup vs. CPU**		
	FPS	1C/1T	1C/2T	32C/64T
HEP-CNN	252.1	9.59	3.3	0.66
CosmoGAN	55.2	2.46	1.58	0.43
3DGAN	35.6	3.10	1.31	0.46

*Arria 10 FPGA at 20 nm process
(FPS: Frames Per Second)
**Intel Xeon Gold 6130 CPU at 14 nm process
(C: cores; T: threads)

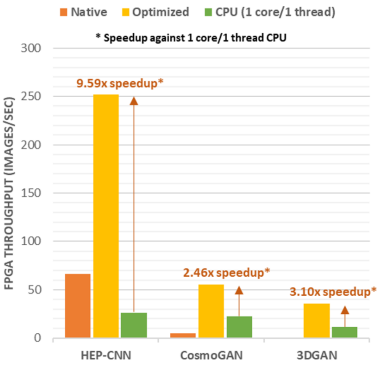

Figure 3: FPGA performance vs. CPU performance

5.1.2 CosmoGAN

As mentioned in Section 4, inferencing CosmoGAN on FPGA requires the implementation of the transposed convolution primitive in the DLA. As detailed in [25], we managed to solve this problem by customizing the DLA to enable the primitive that allows transposed convolution to be inferenced using normal convolution primitive with transposed weights.

However, the FPGA's inference performance of CosmoGAN was still slow comparing to CPU. This is due to a second problem of CosmoGAN that its Normalization layer is implemented to be executed before the Activation layer, while the Normalization module in the DLA is hardwired to be executed after the Activation module (see Fig. 2). This forces DLA to execute these two layers in two separated iterations. To solve this issue, we implemented a custom ReLU Activation module that attaches to the Xbar. We configured DLA to bypass the original Activation module and configured the Xbar to execute the Normalization before the custom ReLU Activation in a single iteration. This customization achieves a speedup of 2.46x against 1 core/1 thread Intel SkyLake CPU (see Table 1 and Fig. 3) which is 12x faster than the native result. The result is also in agreement with our prediction in the paper [25].

5.2 Optimizing 3DGAN

The 3DGAN model cannot be inferenced on the FPGA using the native DLA because the DLA is originally designed for 2D models with 2D convolutional layers. In the native DLA, to inference a 2D convolutional layer, DLA first prepares all input windows and filter data in the host memory. Then, as shown in the middle of Fig. 4, the FPGA kernel loads and vectorizes the data along the input channel (C_{in}) dimension to form $1 \times 1 \times C_{in}$ vectors before storing them into on-chip buffers. Each pair of input and filter vectors are streamed into the PE array module to perform vector-to-vector dot product. The results are accumulated to become the convolution output for the current window while the final output is a concatenated result produced by repeating all above steps for each window.

For the 3DGAN model, however, its 3D convolutional layers have inputs and filters with an extra depth dimension comparing to their 2D counterparts. The 3D convolution window also moves along the depth axis in addition to height and width. To accommodate this behavior, the input/filter vectorization process is modified to take the depth dimension into consideration in the custom DLA (see Fig. 4). To customize the native DLA architecture, before sending

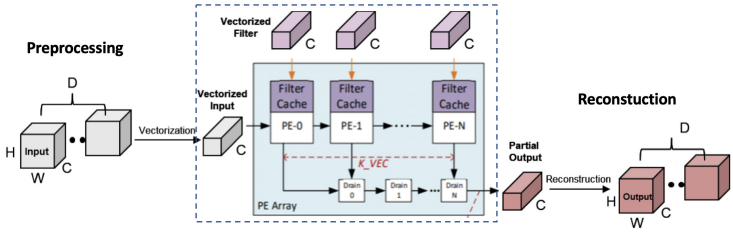

Figure 4: Vectorization process of 3D convolution primitive in custom DLA.

each vectorized input to the PE array, a group of counters are implemented to keep track of their original indices and the position of the current window (see “Preprocessing” in Fig. 4)). Therefore, the PE array (without any change) can use this information to fetch corresponding filter vector from the filter buffer accordingly and stores the dot product outputs based on the position of the current convolution window to reconstruct the final output. Once all counters have reached to their end values, indicating that inputs in the current window have all been fed to the PE array, a "flush_accumulator" flag will be asserted to terminate the accumulation process and send the PE output to the next module (see “Reconstruction” in Fig. 4). For the 3DGAN model, a new counter is added in "Preprocessing" to keep track of the depth index of the vectorized input and the control logic for the "flush_accumulator" flag is also modified to wait for the depth counter to finish before signaling to the PE array.

With the customized DLA, we not only successfully inferenced 3DGAN model with 3D convolutional layers on the FPGA, but also achieved 3.1x speedup against 1 core/1 thread CPU (see Table 1 and Fig. 3).

6 Conclusions and Future Directions

Heterogeneous computing (HGC), using CPUs integrated with accelerators such as GPUs and FPGAs, offers unique capabilities to accelerate DNNs. In this paper, we presented an HGC workflow in performing deep-learning studies on scientific DNN models. In particular, we focused on the use of Intel’s OpenVINO to facilitate the use of FPGA-accelerated inferencing on the HEP-CNN and CosmoGAN models from NERSC (Lawrence Berkeley Lab) and the 3DGAN model from CERN openlab.

From the results presented in Section 5, we demonstrated that that (using customized DLA primitives), for scientifically relevant DNN models such as HEP-CNN and CosmoGAN, a single Arria 10 FPGA (20 nm technology) can produce speedups of 9.59x and 2.46x, respectively, against a single core (single thread) of a server-class CPU (Skylake CPU, 14 nm technology). Using newly developed DLA primitives, we were able to produce a speedup of 3.1x for 3D-GAN. Going forward, from a FPGA device point of view, we are looking forward to working with the PAC card equipped with an Intel Stratix 10 (14 nm technology). From a framework and tools point of view, the lessons learned thus far in using OpenVINO and the DLA development suite will be invaluable in our effort to enhance the DLA primitives and architecture to support existing and emerging scientific DNN models and applications.

Finally, the results from this study, as exemplified by the results presented in Section 5, provide an excellent foundation for more extensive data space exploration going forward to investigate various architectural, model, and tool tradeoffs on performance and other important metrics such as power and cost.

7 ACKNOWLEDGEMENT

This research is funded in part by the NSF SHREC Center and the National Science Foundation (NSF) through its IUCRC Program under Grant No. CNS-1738420; and by NSF CISE Research Infrastructure (CRI) Program Grant No. 1405790.

References

[1] J. Han, J. Pei, M. Kamber, *Data mining: concepts and techniques* (Elsevier, 2011)

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin et al., *TensorFlow: Large-scale machine learning on heterogeneous systems* (2015), software available from tensorflow.org, http://tensorflow.org/

[3] F. Chollet et al., *Keras*, https://github.com/fchollet/keras (2015)

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, arXiv preprint arXiv:1408.5093 (2014)

[5] J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. Zhang, Y. Wan, Z. Li et al., arXiv e-prints arXiv:1804.05839 (2018), 1804.05839

[6] T. Kurth, J. Zhang, N. Satish, I. Mitliagkas, E. Racah, M.A. Patwary, T. Malas, N. Sundaram, W. Bhimji, M. Smorkalov et al., arXiv e-prints arXiv:1708.05256 (2017), 1708.05256

[7] Kurth, Thorsten, *Hep-cnn github repository*, https://github.com/NERSC/hep_cnn_benchmark.git

[8] M. Mustafa, D. Bard, W. Bhimji, Z. Lukić, R. Al-Rfou, J. Kratochvil, arXiv e-prints arXiv:1706.02390 (2017), 1706.02390

[9] S.V. F. Carminati, G. Khattak, Presented at the 23rd international Conference on Computing in High Energy and Nuclear Physics (CHEP 2018). Proceedings in publication. (2018)

[10] *Dell emc ai challenge*, https://insidehpc.com/aichallenge

[11] E. Nurvitadhi, S. Subhaschandra, G. Boudoukh, G. Venkatesh, J. Sim, D. Marr, R. Huang, J.O.G. Hock, Y.T. Liew, K. Srivatsan et al., Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA 17 (2017)

[12] D. Wang, J. An, K. Xu, arXiv e-prints arXiv:1611.02450 (2016), 1611.02450

[13] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran et al., Journal of Instrumentation **13**, P07027 (2018), 1804.06913

[14] Intel, *Openvino toolkit*, https://software.intel.com/en-us/openvino-toolkit

[15] DeePhi, *Deephi dnndk*, http://www.deephi.com/technology/dnndk

[16] T. Sjöstrand, S. Mrenna, P. Skands, Computer Physics Communications **178**, 852 (2008)

[17] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, M. Selvaggi, Journal of High Energy Physics **2014**, 57 (2014), 1307.6346

[18] Wikipedia, *Wikipedia pseudorapidity*, https://en.wikipedia.org/wiki/Pseudorapidity

[19] R.H.R. Hahnloser, R. Sarpeshkar, M.A. Mahowald, R.J. Douglas, H.S. Seung, Nature **405**, 947 (2000)

[20] K. He, X. Zhang, S. Ren, J. Sun, arXiv e-prints arXiv:1502.01852 (2015), 1502.01852

[21] S. Ioffe, C. Szegedy, arXiv e-prints arXiv:1502.03167 (2015), 1502.03167

[22] S. Agostinelli et al. (GEANT4), Nucl. Instrum. Meth. **A506**, 250 (2003)

[23] P. Lebrun, L. Linssen, A. Lucaci-Timoce, D. Schulte, F. Simon, S. Stapnes, N. Toge, H. Weerts, J. Wells (2012), 1209.2543

[24] M.S. Abdelfattah, D. Han, A. Bitar, R. DiCecco, S. OConnell, N. Shanker, J. Chu, I. Prins, J. Fender, A.C. Ling et al., arXiv e-prints arXiv:1807.06434 (2018), 1807.06434

[25] C. Jiang, D. Ojika, T. Kurth, S. Vallecorsa, B. Patel, H. Lam et al., *Acceleration of Scientific Deep Learning Models on Heterogeneous Computing Platform with Intel® FPGAs*, in *International Conference on High Performance Computing* (Springer, 2019), pp. 587–600