2022 International Conference on Futuristic Technologies (INCOFT)
Karnataka, India. Nov 25-27, 2022

Realization of OpenCL based CNN Implementation on FPGA using SDAccel Platform

Abhishek Chaurasiya
Dept. of ESE, National Institute of Electronics and Information Technology.
Aurangabad, Maharashtra, India.
abhichaurasiya19@gmail.com

Abhay Pratap Singh Shekhawat
Dept. of ESE, National Institute of Electronics and Information Technology.
Aurangabad, Maharashtra, India.
abhayshekhawet@gmail.com

Pawan Chaurasiya
Dept. of ESE, National Institute of Electronics and Information Technology.
Aurangabad, Maharashtra, India.
chaurasiyap815@gmail.com

Pawan Kumar Patel
Sr. Technical Assistant,
Dept. of ESE, National Institute of Electronics and Information Technology
Aurangabad, Maharashtra, India.
pawankumar@nielit.gov.in

Prashant Pal
Scientist-'B'
Dept. of ESE, National Institute of Electronics and Information Technology
Aurangabad, Maharashtra, India
prashantpal@nielit.gov.in

Shashank Kumar Singh
Scientist-'B'
Dept. of ESE, National Institute of Electronics and Information Technology
Aurangabad, Maharashtra, India.
shashank@nielit.gov.in

Saurabh Bansod
Scientist-'C'
Dept. of ESE, National Institute of Electronics and Information Technology
Aurangabad, Maharashtra, India
saurabhbansod@nielit.gov.in

2022 International Conference on Futuristic Technologies (INCOFT) | 978-1-6654-5046-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INCOFT55651.2022.10094321

Abstract - **In this paper, we are demonstrating the implementation flow of algorithms using deep Convolutional Neural Networks (CNN). Deploying CNNs solely on CPU-GPU based platforms deal with issues of high power consumption and comparatively slower processing speeds than hardware accelerators like FPGA. FPGA offers high parallel-processing capability, hardware re-configurability, reduced latency and low power consumption. To further improve flexibility in programming the hardware accelerator, we decided to use High Level Synthesis (HLS) tool based on OpenCL framework for development of our CNN model and implementing it on cloud. We are using Xilinx SDAccel development environment targeting Amazon's AWS EC2 F1 instances. We are using Cloud based services for our model visualization on FPGAs. Cloud provides FPGA as a service with High Performance Computing (HPC) resources and enables us to deploy our optimized code on provided FPGA accelerated instance without need of buying an actual hardware. AWS EC2 F1 instance uses Xilinx Ultrascale+ VU9P chip as a Hardware platform.**

Keywords- FPGA, SDAccel, OpenCL, AWS F1 instance, CNN

I. INTRODUCTION

Nowadays, the application of deep Convolutional Neural Network (CNN) is exponentially increasing with the increase in the complexity of the design. For many years, these compute intensive tasks were processed and deployed on CPU- based processors. Since the sustainability of Moore's law is gradually becoming doubtful with increase in number of transistors per chip, there is not satisfactory increase in clock speeds of processors. This acts as bottleneck to get high efficiency and latency using CPU-based architectures. Although GPU-based accelerators are capable in delivering massive parallelism to support neural network deployment, but their high power consumption and not living up to the latency requirements makes them difficult to operate for embedded and IoT applications. In recent times, with growing complexity of neural networks, CPU/GPUs failed to deliver satisfactory results to the developers. Due to several bottlenecks in performance, power efficiency and latency, hardware accelerators like FPGAs proved to be ideal for complex task deployment. FPGAs come into play with their rich computing resources, heterogeneous infrastructure, hardware re-configurability, reduced latency and excellent power efficiency. Traditional Hardware Descriptive Languages (HDL) – Verilog & VHDL would take much long development time for large and complex CNN models which mainly rely on register transfer level (RTL) design. Thus, performing simulation would be time-consuming.

Therefore, we have decided to demonstrate our CNN-based Object Detection (OD) model on FPGA using the OpenCL framework on SDAccel platform. Software defined accelerated development environment leverages on optimization of process so that developers could focus on better algorithm implementation instead of setting-up low-level hardware design functionality. OpenCL framework automatically converts high-level (C/C++) code into low-level RTL code. Using this framework can significantly reduce the development time as algorithm gets divided into multiple kernel codes. Using Cloud based services for FPGA development adds another dimension in our work. Cloud services like Amazon EC2 F1, Intel Cloud offers customizable FPGA instances to accelerate our key workloads. These platforms are useful in accelerating multi-kernel pipelined CNNs delivering higher throughput. We propose to study our model on AWS F1 instance. In recent studies, Cloud Computing proved to be helpful in researches like Genome Sequencing [5,6,7], Real Time Video Processing, Gaming, Augmented Reality, Network Security and Big Data Analytics. Processing massive computation of genomic data on cloud can yield up to 30x better results with accuracy and time than GPU based accelerators. Cloud ensures delivering high performance for applications based on Video Transcoding. A distributed architecture can be created using multiple Hosts - multiple FPGAs in a system.

[8] deploys a CNN model on a distributed Multi-FPGA platform to optimize the MINLP solver algorithm. Our paper is focused more on distributed and heterogeneous cloud infrastructure for development of highly optimized applications. Purely relying on Hardware-based approach would limit the performance of developers due to complexity in dealing with custom hardware accelerators like FPGA. This paper uses Xilinx synthesis tools and development environment which manages the IP cores and runtime allow users to focus towards kernel design efficiently.

In this paper, we are demonstrating Object Detection (OD) using YOLOv2 CNN model on Darknet-19. We are implementing Winograd Convolutional algorithm engine across 3 X 3 CNN layer benchmark suite using 82.1 % of SDAccel resources deploying on Xilinx Ultrascale MultiProcessor System-on-Chip (MPSoC) for heterogeneous computing on AWS cloud.

II. Related Works

In recent times, applications based on neural networks became significant in the field of research. Object Detection, Face Recognition, License number plate detection, Traffic monitoring and Motion detection implemented using multi-layer CNN. Several works have been proposed for acceleration of CNN on FPGA. [1], [2], [3] presented an optimized and quantized CNN structure based on parallel computing approach. [1]-presented an optimized CNN algorithm using Vivado HLS to improve its efficiency and modeling. [2] used a quantized and pruned YOLOv2 model for object detection to implement an efficient CNN model on FPGA accelerator using OpenCL framework. The authors of [2] also presented the latency comparisons between GPU & FPGA by varying the number of Processing Elements (PE) running in parallel. [3]-compare the detection accuracy in terms of mean Accuracy Precision (mAP) and speed (in ms) with different CNN models. In [3], author used Darknet-53 as backbone network. [4] demonstrated YOLOv2 model implementation based on OpenCL framework on FPGA. The author chose Winograd algorithm for CNN acceleration as this algorithm replaces multiplication with addition to speed up the operations.

III. DESCRIPTION OF TOOLS

A. Amazon F1 Instance

AWS F1 instance is a compute instance that allows developers to create custom hardware accelerators for their workloads. It includes up to 8 FPGAs for cloud compute acceleration. Each FPGA in F1 instance provides 2 million+ logic cells, more than 6800 DSP blocks and flexibility to customize instruction set. F1 instances come with Hardware Development Kit (HDK), Software Development Kit (SDK) and reusable Amazon FPGA Image (AFI). Users can deploy compute intensive applications by registering AFI generated by F1 instance. It comes with Xilinx SDAccel environment compatible with OpenCL framework providing development for Host CPU as well as FPGA Kernel. It abstracts I/O using well-defined automated resources so that focus could be shifted on algorithm design.

B. SDAccel

SDAccel is a framework tool created by Xilinx to provide software development support for designers to accelerate their HPC applications using C/C++, OpenCL and RTL programming languages. SDAccel runs on RedHat linux OS in batch mode. Its command line tools enable to add target device, host files and creating kernels. Many Compute Units (CU) can be implemented for each kernel on targeted device and each CU has several Processing Elements (PE) based on MIMD architecture. More the number of CUs instantiated on the device, higher the level of parallelism in the design. SDAccel allows Software emulation, Hardware emulation and Hardware deployment on local host as depicted in Fig.1. It is F1 platform aware too. SDAccel creates development environment for both Host Application on CPU and FPGA Accelerator kernels concurrently as shown in Fig.2. CPU and FPGA device communicate through PCIe using drivers. It provides up to 30 Gigabits per second transfer rates between CPU & DDR through PCIe express and 48Gigabit per second transfer rates between custom FPGA kernel & DDR through AXI interface.

Host Application - C/C++ code is compiled into host executable (.exe) file which gets deployed on host CPU during runtime.

FPGA Kernel - C/C++, OpenCL based hardware accelerator code is compiled into FPGA Binary which gets deployed on Hardware Accelerator.

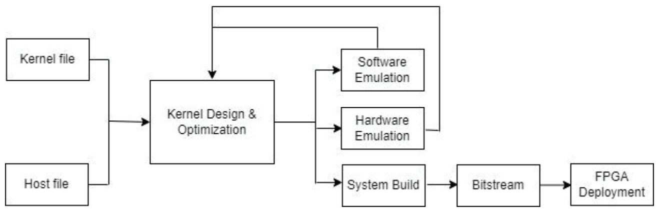


Fig. 1. SDAccel Design Flow

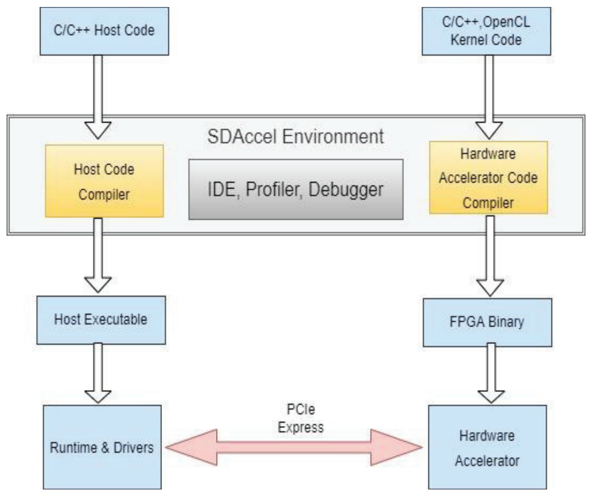


Fig. 2. . SDAccel Working Flow

C. OpenCL

Open Compute Language is a platform independent heterogeneous programming model for parallel computing for CPUs, GPUs and FPGAs. This synthesis tool provides code optimization and OpenCL Runtime to manage communication with kernels. SDAccel provides kernel development in OpenCL and OpenCL APIs on host side to transfer and manage computing tasks. OpenCL device side transfers data from DDR to Global Memory and then communicates with the Host side through PCIe express as shown in Fig.3.

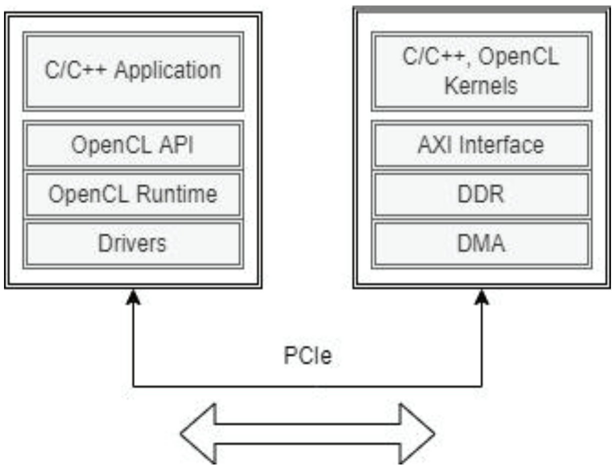


Fig. 3.

IV. CNN ARCHITECTURE

CNN is a deep learning algorithm which takes image as an input which is pre-processed to detect the object using feature extraction and detection Fig.( ) shows the complete CNN process. Firstly, the image matrix is convolved with the appropriate filters (3*3 in our case). This step filters the image into different set of neurons as feature detectors to perform feature map. Filters can used for edge detection, face detection, body detection, etc. *Relu* Activation function is applied on each field in CNN filtered kernel output. *Padding* (P) is provided in image matrix to reduce pixel loss in feature detector matrix. Pooling layer further reduces the dimensional size of the image using *Max Pooling* filter. Pooling reduces computation, over fitting and tolerant towards variations and distortions. It selects high intensity or low intensity pixels to further differentiate specific features in an object. The process of conv + pooling continues to detect features and their sub-features. Many conv layers can be stacked horizontally for necessary feature extraction operations. On each step, CNN model learns the filters and update their values with the help of *back propagation*. After feature detection, a grid is formed for classification of the detected features. It creates a dense neural network resulting in a classified image as an output.

A. Winograd Convolution

It is based on Winograd minimum filtering technique where Multiplication operations are reduced into Addition operations to reduce computational calculations. As the convolution layer's computations take a long time and need a lot of processing resources because of the complex calculations and substantial quantity of data involved. The computation of the convolution layer is hence the primary emphasis of this study. In general, input feature kernels, maps, and extracted feature maps are the standard inputs and outputs data for such convolution layer. The output feature maps include the results of the convolution operation between the input feature maps and convolution kernels, which is subsequently applied to the following layer. Assume that the input feature map has N pieces and is H W in size. The size of the convolution kernels is (N x K x K), and there is M of them. The number of components in the feature maps input must match one of the dimensions within every convolution kernel. Each feature map's allotted kernel size for the convolution process is K X K. If the stride is S, each K X K kernel performs one operation for each of the N feature maps before adding the results to create a unit component of the output feature map. As a result, if the preceding layer's M convolution kernels, the following layer's M feature mappings must also be M. R=H-KS+1 and C=W-KS+1 if the dimension of the output feature map is R x C.

$$Out[M][R][C] = \sum_{n=0}^{N}\sum_{i=0}^{K}\sum_{j=0}^{K} In[n][S\times r+i][S\times r+j]\times W[M][n][i][j] \qquad (1)$$

With a reduced convolution kernel size, the Winograd minimum filtering technique significantly reduces the amount of work required for convolution. The convolution kernels for the YOLOv2 algorithm are all 3X3 and 1X1. These can be used with the Winograd technique to speed up the convolution process because they are suited for usage at such small sizes. The Winograd approach determines the convolution kernel F(m,r) having m outputs and dimension by multiplying v(F(m,r))=m+r1. Equation (2) shows that when the convolution kernel size is three components and the output matrix is two dimensions, the Winograd minimum filtering technique is used to the convolution process. The output data is represented by mi, the convolution kernel data is represented by gi, and the input data is represented by di.

$$F(2,3) = \begin{bmatrix} d0 & d1 & d2 \\ d1 & d2 & d3 \end{bmatrix}\begin{bmatrix} g0 \\ g1 \\ g2 \end{bmatrix} = \begin{bmatrix} m0+m1+m2 \\ m1-m2-m3 \end{bmatrix} \qquad (2)$$

$$m0 = (d0-d2)g0$$
$$m1 = (d1+d2)\frac{g0+g1+g2}{2}$$
$$m2 = (d0-d1)\frac{g0-g1+g2}{2}$$
$$m3 = (d1-d3)g2$$

The Winograd minimum filtering technique produces a vector of m dimensions from picture data with an input of mr+1 pixels.

The output matrix size in the 2D convolution computation is m x m and the convolution kernel size is indicated by F(m m,r r). The dimension of the input picture data must be (m+r1) (m+r1) when the output matrix is m x m in size. Thus, we can determine the values of m1, m2, m3, and m4. Then instead of calculating the dot product of matrices, utilize them to perform convolution. We may note that since the filter is constant, the values of (g0 + g1 + g2) / 2 and (g0-g1 + g2) / 2 do not need to be determined for each convolution. They can be pre-calculated once during training before convolution and kept for use during inference. The values of m1, m2, m3, and m4 must now be calculated using 4 ADD and 4 MUL operations. Then, use 4 ADD operational processes to calculate the result using the computed values of m1, m2, m3, and m4. We would do six MUL operations rather than four for creating standard dot products. This approach increases the number of addition operations by adding the input data four times, the convolution kernel three times and the multiplied data four times. However, there are only four multiplications rather than the usual six. This significantly reduces the computationally intensive MUL operation by 1.5 times.

B. YOLOv2

To reflect the acceleration effect of FPGAs, we are using the YOLOv2 network of the YOLO series. The framework used by YOLOv2 is the improved Darknet-19. This model is trained on a COCO image dataset containing 80 image classes. 19 convolution layers, 1 Average pooling layer, 5 Maximum pooling layers, and one1 Softmax layer make up the Darknet-19 network. In Darknet-19, there are a lot of 3X3 convolution filters. A 1X1 convolution filtering is added between each of the 3X3 convolution filters to compress information and enhance network depth. The final convolution, the average pooling layer, and the softmax layer are eliminated, three 3X3 convolution layers and one 1X1 convolution layer are added, and convolution is used in place of a fully connected layer to enhance YOLOv2. The YOLOv2 model on integrated terminals takes longer to execute due to a lot of convolution operations. However, the sequence of convolution operations in YOLOv2 may be carried out by the FPGA's high-efficiency parallel computing capacity. When we created the FPGA accelerator, the Winograd algorithm was added to the YOLO network's convolution processes.

V. PROPOSED METHODOLOGY

Our work makes use of Xilinx SDAccel IDE to deploy applications targeting Amazon AWS F1 instance which offers FPGA as a service on cloud. Fig.2 shows the working flow of SDAccel environment consists of an IDE, Profiler for application optimization and a Debugger. Flow diagram consists of two sections – (1) Host CPU and (2) FPGA kernels.

The host CPU creates our CNN C++ code for object detection in host file using SDAccel environment. Input image is obtained from Host side application. Host manages the device by instantiating kernels and creating buffers. The input image gets pre-processed and application code is executed and complied with FPGA kernels on device. Memory partitioning for optimal usage of BRAM resources, DSP block inference, loop unrolling, loop pipelining optimizations are done on device side by SDAccel. The SDAccel platform then profile, debug, deploy and validate the process on F1 instance. OpenCL automates the execution flow and converts it into Binary for FPGA deployment. SDAccel automate the operations to read, write, process data in DDR using AXI Interface. It synthesizes the kernel code and performs necessary placing & routing for FPGA logic cells. Once FPGA Binary Bitstream file is generated after compilation, AWS creates an encrypted Amazon FPGA Image (AFI). SDAccel allow users to deploy the accelerated application on AWS cloud F1 instance or on premises (local host). Fig.4 depicts the development flow from launching in to AWS and selecting the instance pre-loaded with FPGA Developer Amazon Machine Image (AMI). Users can select up to 8 FPGAs per instance with 64 GB DDR4 protected memory and a dedicated PCIe x16 connection to an instance. SDAccel environment let users to connect host side CPU with multiple FPGA instances to ensure more parallelism. Fig.5 shows the types of f1 instances available on AWS.

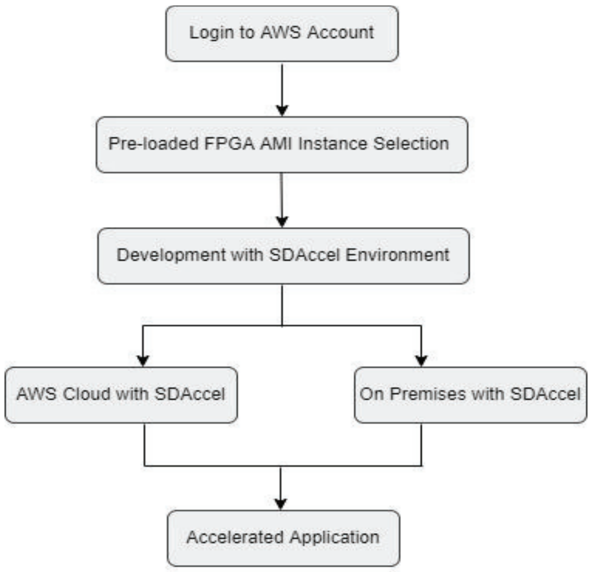
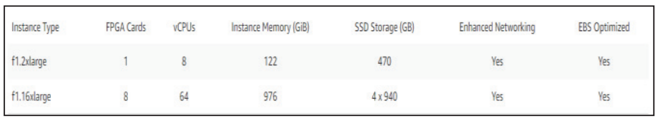


Fig. 4.



Instance Type	FPGA Cards	vCPUs	Instance Memory (GiB)	SSD Storage (GB)	Enhanced Networking	EBS Optimized
f1.2xlarge	1	8	122	470	Yes	Yes
f1.16xlarge	8	64	976	4 x 940	Yes	Yes

Fig. 5. Source: AWS

VI. RESULTS

We use OpenCL on SDAccel environment to perform our YOLOv2 CNN implementation on Darknet-19. Our design is deployed Amazon EC2 F1 instance with working frequency of 212MHz. Fig.6 shows the experimental result of the accelerated application with person & a bottle detected. The total resource utilization after place and route is shown in Table I. Our implementation result shows the data throughput of 550 FPS with power usage of 35 Watts with Accuracy of 72% and Efficiency of 32GFLOPs/Watt. We also compared our result in Table II with CPU implementation on Intel i-5, 12 GB RAM and NVIDIA 3010 TI processor.

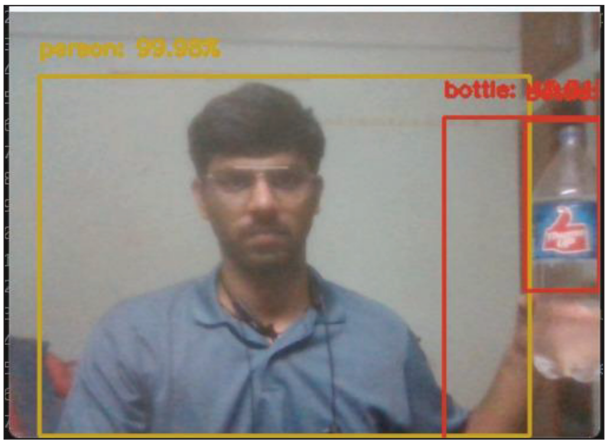


Fig. 6.

TABLE I.

Resource Parameters	FF	LUT	DSP	BRAM
Ultrascale+ Total Resources	866,400	433,200	4,200	2,940
SDAccel Region Resources	549,050	275,520	2,576	1,940
Winograd Conv. Resources	249,851	229,226	1,507	1,288

TABLE II.

Parameters	**FPGA**	**CPU**
Throughput (FPS)	550	55
Power (Watt)	25	120
Efficiency (GFLOPS/Watt)	42	2.2
Accuracy (%)	74	69
Latency (ms)	9	19

VII. CONCLUSION

We have demonstrated a CNN implementation of Object Detection technique using OpenCL on SDAccel and accelerated our optimized application on AWS EC2 F1 instance using Virtex Ultrascale+ FPGAs. Using the proposed methodology, many CNN architectures can be implemented on any network for several HPC applications.

REFERENCES

[1] L. Wen, "FPGA-Based Deep Convolutional Neural Network Optimization Method," 2021 International Conference on Signal Processing and Machine Learning (CONF-SPML), 2021, pp. 109-112, doi: 10.1109/CONF-SPML54095.2021.00030.

[2] A. Yang et al., "An OpenCL-Based FPGA Accelerator for Compressed YOLOv2," 2019 International Conference on Field-Programmable Technology (ICFPT), 2019, pp. 235-238, doi: 10.1109/ICFPT47387.2019.00036.

[3] P. Babu and E. Parthasarathy, "Optimized Object Detection Method for FPGA Implementation," 2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2021, pp. 72-74, doi: 10.1109/WiSPNET51692.2021.9419407.

[4] C. Cui, F. Ge, Z. Li, X. Yue, F. Zhou and N. Wu, "Design and Implementation of OpenCL-Based FPGA Accelerator for YOLOv2," 2021 IEEE 21st International Conference on Communication Technology (ICCT), 2021, pp. 1004-1007, doi: 10.1109/ICCT52962.2021.9657856.

[5] D. Fujiki et al., "SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 937-950, doi: 10.1109/MICRO50266.2020.00080.

[6] J. Cong, "Characterization and acceleration for genomic sequencing and analysis," 2017 IEEE International Symposium on Workload Characterization (IISWC), 2017, pp. 1-1, doi: 10.1109/IISWC.2017.8167744.

[7] Z. Huang, J. Yu and F. Yu, "Cloud processing of 1000 genomes sequencing data using Amazon Web Service," 2013 IEEE Global Conference on Signal and Information Processing, 2013, pp. 49-52, doi: 10.1109/GlobalSIP.2013.6736809.

[8] J. Shan, M. T. Lazarescu, J. Cortadella, L. Lavagno and M. R. Casu, "CNN-on-AWS: Efficient Allocation of Multikernel Applications on Multi-FPGA Platforms," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 2, pp. 301-314, Feb. 2021, doi: 10.1109/TCAD.2020.2994256.