IEICE TRANS. INF. & SYST., VOL.E106–D, NO.7 JULY 2023

PAPER

Parallel Implementation of CNN on Multi-FPGA Cluster

Yasuyu FUKUSHIMA[†a)], **Kensuke IIZUKA**[†], *Nonmembers*, *and* **Hideharu AMANO**[†], *Fellow*

SUMMARY We developed a PYNQ cluster that consists of economical Zynq boards, called M-KUBOS, that are interconnected through low-cost high-performance GTH serial links. For the software environment, we employed the PYNQ open-source software platform. The PYNQ cluster is anticipated to be a multi-access edge computing (MEC) server for 5G mobile networks. We implemented the ResNet-50 inference accelerator on the PYNQ cluster for image recognition of MEC applications. By estimating the execution time of each ResNet-50 layer, layers of ResNet-50 were divided into multiple boards so that the execution time of each board would be as equal as possible for efficient pipeline processing. Owing to the PYNQ cluster in which FPGAs were directly connected by high-speed serial links, stream processing without network bottlenecks and pipeline processing between boards were readily realized. The implementation on 4 boards achieved 292 GOPS performance, 75.1 FPS throughput, and 7.81 GOPS/W power efficiency. It achieved 17 times faster speed and 130 times more power efficiency compared to the implementation on the CPU, and 5.8 times more power efficiency compared to the implementation on the GPU.

key words: *FPGA, multi-FPGA, MEC, CNN*

1. Introduction

Recent research has revealed that deep learning can achieve high recognition accuracy in image recognition and natural language processing. Moreover, it can be applied to various fields, such as automatic driving, automatic translation, and medical care. In particular, convolutional neural networks (CNNs) have high recognition accuracy in the field of image recognition. However, a CNN requires a large calculation amount, and problems of increased latency and power consumption occur when the CNN is executed on a general-purpose processor. To address these issues, a domain-specific architecture (DSA) implemented on a field-programmable gate array (FPGA) is a promising candidate.

FPGAs can maintain low development costs, shorten the development period compared to dedicated chips such as ASICs, and consume less power than GPUs. Owing to these advantages, they are used not only for conventional IoT and edge devices, but also for a computing infrastructure installed at the base station of a 5G mobile network that realizes multi-access edge computing (MEC). However, high-end FPGAs, which provide powerful computing functions and enormous on-chip memory, tend to be expensive. In addition, even in such a high-end FPGA, there are limits to available resources with a single FPGA. To address this problem, we developed a PYNQ cluster which consists of cost-efficient Zynq boards, called M-KUBOS [1], connected with low-cost, high-performance GTH serial links. The PYNQ Linux-based open-source software environment was introduced for multi-tenant processing on a multi-FPGA system. Multiple timing-critical applications in MEC can be executed on multiple boards of the PYNQ cluster in parallel.

In this study, as an example of image recognition for MEC application, ResNet-50, a typical CNN, was implemented on the PYNQ cluster. The performance and power consumption of the actual system were evaluated. The purpose of our work is to indicate below two points by demonstrating that ResNet-50 can be executed with high performance and high power efficiency on a PYNQ cluster. (1) Implementing a CNN inference accelerator on a multi-FPGA system is a promising option. (2) The PYNQ cluster has sufficient computing power as a MEC server.

The remainder of this paper is organized as follows. Section 2 describes MEC and PYNQ clusters. Section 3 describes CNNs and ResNet. Section 4 introduces research related to our work. In Sect. 5, division and parallelization methods based on the computational characteristics of CNNs are proposed, and the FPGA design and implementation are described. In Sect. 6, the performance of the implementation is evaluated, and Sect. 7 concludes this paper.

2. Background

2.1 MEC

MEC is being standardized by the European Telecommunications Standards Institute (ETSI) for 5G mobile networks. MEC provides an IT service environment and cloud computing functions for users at the edge of an access network that includes multiple access technologies. Although the MEC background objective is an advancement of 5G wireless technology, the same concept holds not only in 5G but also in WiFi, low-power wide-area (LPWA), or wired network environments. The advantages of MEC are summarized as follows. (1) Tasks can be offloaded from an edge device to a nearby MEC. (2) The application can be operated in the local environment, thereby improving the response time and user convenience. (3) It is possible to save network bandwidth with the cloud and reduce network con-

Manuscript received September 26, 2022.
Manuscript revised February 8, 2023.
Manuscript publicized April 12, 2023.

[†]The authors are with Dept. of Information and Computer Science, Keio University, Yokohama-shi, 223–0061 Japan.
a) E-mail: fic@am.ics.keio.ac.jp

DOI: 10.1587/transinf.2022EDP7175

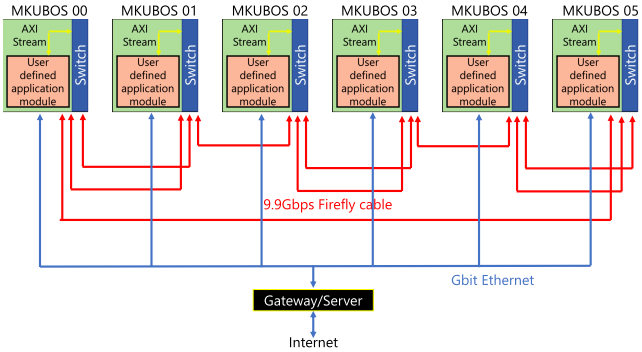


Fig. 1 Interconnection of PYNQ Cluster.

gestion. With the use of these features, it is expected that various applications, such as factory control, smart city traffic management, and security control are operated on MEC servers. In addition, it is possible to use MEC by offloading heavy tasks that have been performed inside individual terminals to a nearby MEC and executing them. Because it is fully conceivable to use image processing by deep learning for these applications, MEC must support deep learning.

2.2 PYNQ Cluster

FPGA computing is attracting attention as a promising candidate for the MEC server because of its low cost and low power consumption. Another important advantage of FPGA computing is that it can handle timing-critical jobs by accepting a request from an edge device directly through its I/O and by executing jobs with continual performance with hard-wired logic. Multi-FPGA systems are particularly advantageous because they can handle multiple requests from multiple edge devices at a base station.

For the MEC computing platform, we developed a PYNQ cluster [2] consisting of multiple Zynq Xilinx SoC boards equipped with an ARM core called a processing system (PS) and an FPGA called programmable logic (PL). Python productivity for Zynq (PYNQ) [3] is an open-source software platform developed by Xilinx. It is built on Ubuntu Linux and can build a software stack for common Linux servers.

The current PYNQ cluster was built by connecting six M-KUBOS boards from PALTEK [1], as shown in Fig. 1. A photograph of the actual system is shown in Fig. 2. The details of each M-KUBOS board are listed in Table 1. As shown in Fig. 3, the largest size of Zynq, XCZU19EG, is mounted with two DDR4 DRAM sets and several high-speed serial links. A duplicated ring network with five 5-input by 5-output static time-division multiplexing (STDM) switches was formed [4], which can guarantee the bandwidth and latency between boards. A user-defined application module in Fig. 1 is connected to the port of the switch with the 170-bit width AXI Stream interface. Xilinx Aurora IP is used for sending/receiving serial data through the GTH serial cable bundled with Samtec Firefly cables. As shown in Table 1, including the error-correcting code (ECC), the

Fig. 2 Appearance of PYNQ Cluster.

Table 1 Key specifications of M-KUBOS.

Item	Specification
FPGA	XCZU19EG-2FFVC1760
Memory	PS: 4GB DDR4-2400 PL: 1x 4GB DDR4-2400 SODIMM Socket
I/O	4x GTH 8TX (max 16.3Gbps) (effective 8.5Gbps) 4x GTH 8RX (max 16.3Gbps) (effective 8.5Gbps) USB3.0 × 1 USB-UART × 1 1Gb Ether(RJ45) DP1.2

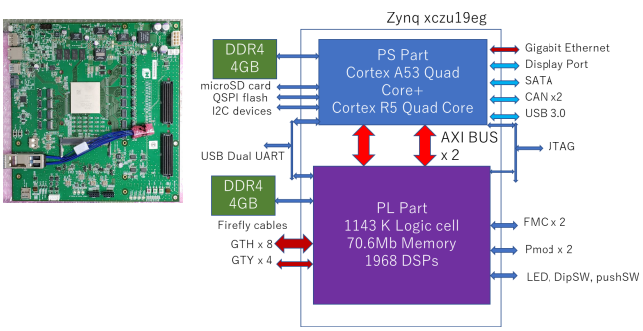


Fig. 3 M-KUBOS board.

effective bit rate is 8.5 Gbps for each link and the total bandwidth between boards is 34 Gbps. Because the PYNQ cluster provides a RESTful API [2], we can configure the PL from the host server by using this API.

As shown in Fig. 4, the user describes JSON-based configuration files to specify the PS. The job is distributed to multiple boards of M-KUBOS with a cluster management system *mkubosmgr* [2], [5]. On each Linux running of the PS of M-KUBOS board, *mkubosmww* [2], [5] manages distributed jobs according to the descriptions. As with other application programs, the implemented ResNet-50 was distributed using the above management system. Although the

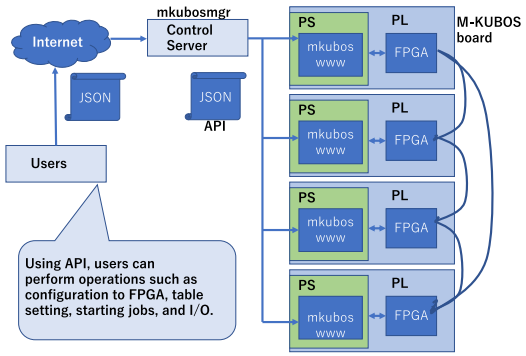


Fig. 4 The way of using PYNQ Cluster.

execution of this paper is evaluated by the manual job distribution, now, the M-KUBOS resource manager is available, and it selects unused boards and dispatches the job automatically by using a cluster management tool Slurm [6]. The detail of the system is the same as that for FiC described in [7].

Although image processing with a CNN is a major application of MEC, introducing dedicated computational components, such as GPUs, is not desirable owing to the limited resources and power for MEC. Zynq is not a high-end FPGA device for computation; it can support sufficient computational power by parallel processing. Here, we investigate the parallel processing techniques of the CNN on the PYNQ cluster.

3. Convolutional Neural Network

3.1 Basics of CNN

A CNN is a type of neural network that exhibits high accuracy in image recognition and object detection. Thus, it is used in many services, such as obstacle detection during autonomous driving. The basic structure of the CNN is shown in Fig. 5. Each layer indicated by an arrow performs an operation on the input data from the previous layer and generates output results that are passed to the next layer.

Because the input/output in the middle of the CNN is the extraction of the feature points of the image, these intermediate results are called feature maps. In particular, the input of each layer is called the input feature map, and the output is called the output feature map. In many layers, the input feature map is calculated using weights and biases to generate the output feature map. In addition, as shown in Fig. 5, each dimension of the 3D feature map is the width, height, and channel, respectively. The CNN layer includes convolution layers, fully connected layers, pooling layers, and activate functions like ReLUs. Among them, the computational complexity of the convolution layer accounts for a large part of the whole.

3.2 ResNet

ResNet [8], which is the target of advancement in our work,

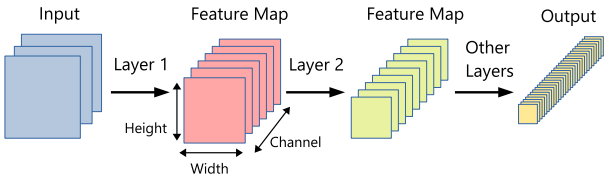


Fig. 5 Pattern diagram of CNN.

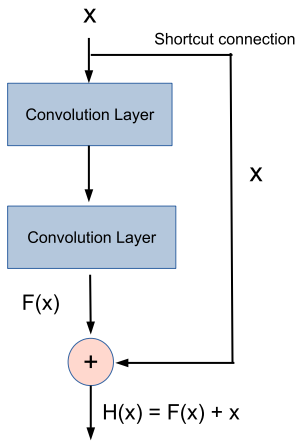


Fig. 6 Basic structure of ResNet.

is a CNN model that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015. It is characterized by a deeper model and more layers than previous models. ResNet has a basic structure called a residual block that has a shortcut connection that skips several layers, as shown in Fig. 6. By stacking this structure over multiple layers, the number of layers can be increased to improve the accuracy by preventing the disappearance of the gradient.

Regardless of the layer depth, ResNet was designed to have a convolution layer and a max-pooling layer. Then, an average-pooling layer, and a fully connected layer are placed at the end. Between them, many residual blocks are provided. ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152 are commonly used, although the number of layers can be further increased. In this study, we implemented ResNet-50 as a typical CNN model with a large number of parameters and a large number of calculations. ResNet-50 was selected as a benchmark for MLPerf [9] and as a target for BrainWave [10]. The structure of ResNet-50 is presented in Table 2.

4. Related Work

Numerous studies have been conducted on the implementation of CNN on FPGAs. This section introduces the works related to our work.

4.1 CNN Implementation on Single FPGA

There are two methods for FPGA-based CNN implementation: the recurrent structure and the pipeline structure. The recurrent structure is a method similar to that in [11],

Table 2 Structure of ResNet-50.

	conv1	max pool	conv2_x	conv3_x	conv4_x	conv5_x	average pool	1000-d fc
output size	112×112	56×56	56×56	28×28	14×14	7×7	1×1	1×1
block	$7 \times 7, 64$	$3 \times 3, 64$	$\begin{pmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{pmatrix} \times 3$	$\begin{pmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{pmatrix} \times 4$	$\begin{pmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{pmatrix} \times 6$	$\begin{pmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{pmatrix} \times 3$	-	-
operations per block	118.0M	1.8M	462.4M	411.0M	411.0M	411.0M	25.1K	513.0K
params per block	9.4K	-	147.5K	524.3K	2097.2K	8388.6K	-	513.0K

which makes full use of on-chip resources to build a unified computing unit and share the unit between different layers. Many CNN accelerators using FPGAs in previous studies, such as [12], are often designed to have only one convolution layer processor (CLP). On the other hand, the pipeline structure implements each layer of the neural network as a separate pipeline stage, as in [13]. With this method, it is possible to implement the optimum design for each layer, but the resources available to each layer are limited. In our implementation, we specially designed the first convolution layer and designed a common recurrent structure for other residual blocks.

4.2 Mapping Approach for CNN on Multi-FPGA

The large size of the deep learning network model tends to lead to resource shortages for implementing accelerators using a single FPGA, and the overall computing performance tends to be limited. To address this problem, researchers have implemented a CNN on multiple FPGAs.

In [14], an efficient search algorithm was proposed for the design space exploration of multiple FPGAs. The execution results of the system designed using the method were up to approximately 21 times and twice the power efficiency of the CPU and GPU in the system consisting of six FPGAs using AlexNet and VGG-16 as the benchmarks. Note that AlexNet and VGG-16 are smaller models than ResNet-50.

In [15], based on the research of [14], the division was performed by an algorithm that also considered the topology of the connection network between FPGAs. The policy of its design space exploration was dynamic programming, as in [14]. However, in [15], FPGA resources were further finely allocated by a binary search. ResNet-152 was implemented and evaluated by changing the number of FPGAs. By using up to 16 FPGAs, the performance improvement was double that of the GPU. Since [15] focuses on mapping problems, the DSP usage rate is around 20%, which is not sufficiently accelerated.

Our design shares common attributes with those of the above studies, specifically, it divides the network model between layers and maps it to a multi-FPGA system. As in our study, ResNet was used for an evaluation in [15] and it is thus easy to conduct a comparison. The difference, however, was the implemented multi-FPGA system. In [14], FPGAs were connected by SATA; hence, communication between FPGAs was not fast. In [15], FPGAs were not directly connected; consequently, communication between FPGAs had to be managed by the host CPU. On the other hand, in our PYNQ cluster, FPGAs are directly connected by high-speed serial links, which enable stream processing without network bottlenecks and facilitate pipeline processing between boards.

An accelerator for the training of CNNs on multi-FPGA system is proposed [16]. This work improves the training performance by using up to 15 FPGAs and eliminates the parameter fetching from off-chip by efficiently distributing the parameters of medium-sized CNN models, AlexNet, VGG-16, and VGG-19 across many FPGAs. On the other hand, ResNet-50 targetted in our work has a larger number of the parameters than medium-sized models targetted in this work, so the parameters are retained on-chip unless extreme quantization such as binarization reduces the recognition accuracy.

To summarize the novelty of our work, there is a difference between these related works in terms of a design with high DSP utilization while holding the parameters in DRAM for the inference process of a large-scale CNN model. Furthermore, our work has the significance of showing that the PYNQ cluster has sufficient computing power as a MEC server.

This paper is an extended version of [17]. Implementations on 2, 3, 5, and 6 boards are added, and there is a discussion about performance and scaling. The power evaluation results from the real system are also added. As a result, the evaluation results are totally updated.

5. Implementation

This chapter describes the design guidelines, the dividing method, and the flow of implementation and execution. After that, quantization and parallelization for speeding up are described.

5.1 Design Guidelines and Dividing Method

Layers of ResNet-50 are divided and implemented on multiple M-KUBOS boards. For example, we explain the case of dividing ResNet-50 into four boards. The processing on each board is a stage of the pipeline, as shown in Fig. 7. Assuming that boards 0 to 3 are used, data are inputted from the PS to the PL of board 0, the processing is performed in order from board 0 to board 3 in the PL, and the intermediate result is passed to the next board. The final result is outputted to the PS of board 3.

In pipeline processing, the overall performance is determined by the stage with the longest execution time. So we should allocate the layers to the FPGA boards properly to make the execution time of each board as equal as pos-

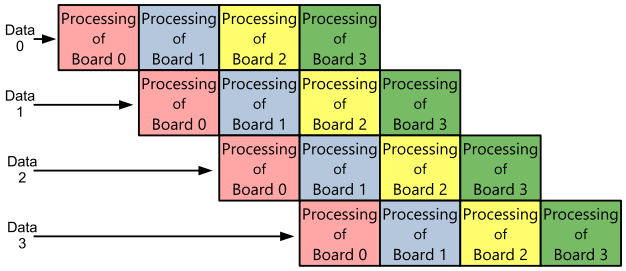


Fig. 7 4 board pipeline processing.

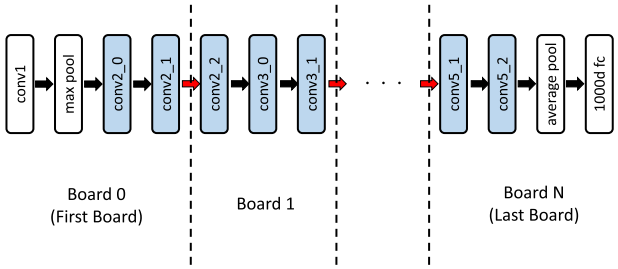


Fig. 8 Example of dividing and implementing ResNet-50 on multiple FPGA boards.

sible. We implemented a common module for convolution layers to simplify the design and allow estimation of execution time before the division of ResNet is determined. All of the residual blocks, such as conv2_x to conv5_x in Table 2, are implemented by using this convolution layer module of common design. The detail of this module is described in Sect. 5.4. Using the same module instead of preparing individual designs for each layer, eliminates the need to consider resource allocation when allocating each layer separately to the FPGA board. However, the shape of the feature map of the first convolution layer and the last FC layer are different from other convolution layers in ResNet. Thus, it is necessary to prepare individual designs. These layers are designed to save resources so that they can be mounted on the same board with the convolution layers of common design. In addition, a max-pooling layer and an average-pooling layer consume fewer resources and can be mounted on the same board with the other layers without any problems. The convolution layers that make up the majority of the ResNet are implemented by using the common design module iteratively. With the above design, any layer can be allocated to the M-KUBOS board without running out of resources. Since the distribution of layers can be determined without resource constraints, only execution time is taken into account when deciding on the division of ResNet.

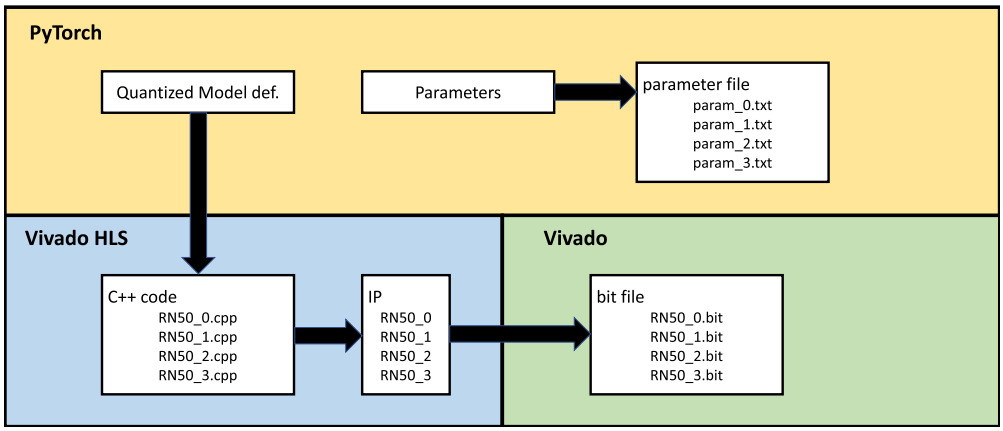
The distribution of layers is determined by the following procedure. To make the processing time of each board as equal as possible, first, high-level synthesis is performed by Vivado HLS by Xilinx, and where we should divide the network is determined from the estimation of the execution time of each layer. After that, it is executed on the real PYNQ cluster to measure the execution time, and the division is fine-tuned. This adjustment is necessary because the communication with the DRAM is likely to make a difference between the Vivado HLS estimation and the real machine behavior. Figure 8 shows an example of dividing and implementing ResNet-50 on multiple FPGA boards. Layers shaded in blue indicate that they are implemented with convolution layers of common design. Also, the red arrows indicate that board-to-board communication takes place.

5.2 Implementation and Execution Flow

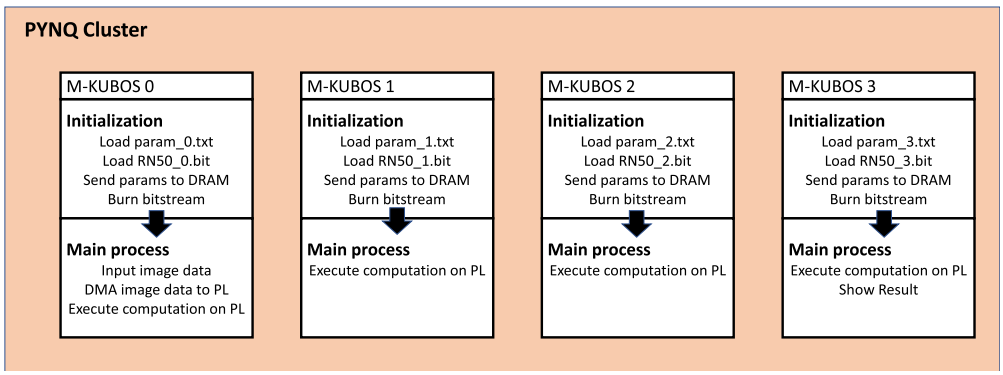
Figure 9 shows the procedure from implementation to execution. This example describes the case of running the ResNet-50 accelerator on 4 boards. First, we get the pre-trained quantized model and the parameters from PyTorch. The parameters are sorted according to the order in which they are used, then saved in text files. Each text file holds the minimum parameters needed to run on each board. Next, we rewrite the model definition in C++ to improve the performance and generate the accelerator IP by using Vivado HLS. The generated accelerator IP is synthesized with the shell part of M-KUBOS which contains the STDM switches and DRAM controller by Vivado and the bitstream files for the PL part of M-KUBOS are generated. This is the end of the implementation process.

The shell part of M-KUBOS and interfaces are designed as follows. Input data are packed into 256-bit width data and transferred with DMA from the PS to PL on board 0 using the Xilinx DMA IP. The data passed between boards are transferred in 170-bit width data to the STDM switch through the AXI4-Stream interface and Aurora IP. As a result, we can connect modules between multiple boards by extending the block design for a single board. In this implementation, 128 bits out of 170 bits are used for the data part; the other parts are necessary for transmission and reception with the STDM switch. Quantization from a 32-bit floating-point to an 8-bit integer is performed for feature maps and weights; thus, sixteen 8-bit integer numbers are sent and received together in the 128-bit width of the data part. In addition, DDR-4 SDRAM on each board is used for holding weight data.

To execute our ResNet-50 accelerator on the PYNQ cluster, initialization in Fig. 9 is required. First, the host program is executed on each board and the corresponding text file and bit file are loaded. Next, the parameters read from the text file are sent to DRAM, and the bit file is sent to PL to complete the initialization. After the initialization is completed, the main process can be operated. On each board, computations on PL are executed. Especially, on the first board, the input data is transferred to PL by DMA. Each board sends intermediate results to the next board after it has finished processing on its allocated layers. The last board displays the inference result on the command line. The main process flow is also shown in Fig. 7. The main process can run repeatedly once it has been initialized. The bitstream file and DRAM are not rewritten while the image recognition task is executed in the main process.



(a) Implementation flow.



(b) Execution flow.

Fig. 9 Procedure from implementation to execution.

5.3 Quantization

As mentioned above, in this implementation, the model trained with 32-bit floating-point numbers was quantized to 8-bit integer numbers for both the weight and feature map. The quantization method is described in [18]. Quantization can reduce the requirements for memory bandwidth and on-chip memory resources. While the multiplication of 32-bit floating-point numbers requires three DSP units, the multiplication of 8-bit integer numbers can be performed by one DSP unit; thus, the DSP unit of the FPGA can be saved with quantization. Moreover, if an 8-bit integer is used for both the weight and feature maps, two MAC operations can be performed with one DSP [19].

We examined the accuracy between this implementation and the floating-point model, and the results are shown in Table 3. Inference by ResNet-50 was performed using 1000 images of ImageNet, and the recognition accuracies of Top-1 and Top-5 were compared. Top-1 and Top-5, which show whether the correct class is included in the top 1 or 5 of the inference results, respectively, are often used to evaluate the recognition accuracy of CNNs. Float32 in Table 3 is the original floating-point model, and INT8 is the precision of this implementation quantized to an 8-bit integer. As a result, it was confirmed that there is almost no decrease in accuracy with quantization.

Table 3 Comparison of recognition accuracy between quantization result and the floating point model.

	Float32	INT8
Top-1 Accuracy	74.4%	74.2% (-0.2%)
Top-5 Accuracy	93.0%	92.8% (-0.2%)

5.4 Parallelizing Convolution Operations

Here, we show the method used to speed up the convolution operation in ResNet-50. In the explanation, the following symbols are used: ICH is the number of channels in the input feature map, IH is the height of the input feature map, IW is the width of the input feature map, OCH is the number of channels in the output feature map, OH is the height of the output feature map, OW is the width of the output feature map, and K is the kernel size. The convolution operation generates the output feature map $OH \times OW \times OCH$ from the weights $OCH \times ICH \times K \times K$ and the input feature map $IH \times IW \times ICH$. The amount of calculation is expressed in Eq. (1).

$$\text{conv\_operations} = OH \times OW \times K \times K \times OCH \times ICH \quad (1)$$

Speeding up is achieved by executing this operation

in parallel on the FPGA. Parallelization is achieved by unrolling the input/output channel loops, as shown in [12]. Because the number of input/output channels of the convolution layers other than the first convolution layer are powers of 2, the unrolling factor is selected by dividing the input and output channels by powers of 2. The number of input channels and output channels is 64 or more in convolution layers other than the first convolution layer of ResNet-50. Because the convolution layers other than the first convolution layer are based on a common design, it is possible to achieve the maximum degree of parallelism of 64 in the calculation. The degree of parallelism is limited by the computational resources of the FPGA board used in this implementation. Here, the kernel loops are not unrolled because sufficient parallelism can be ensured only by unrolling the input/output channel loops.

The design space exploration for determining the unrolling factor of the input/output channel is shown in Algorithm 1. In this algorithm, To and Ti are alternately doubled unless $To \times Ti$ does not exceed twice the number of available DSPs. Note that it is possible to perform two MAC operations with one DSP by 8-bit quantization. Accordingly, the input channel unrolling factor Ti and the output channel unrolling factor To are obtained, and $To \times Ti$ multiplications are executed in parallel.

Algorithm 1 Design space exploration

```
Set available DSP: DSP_NUM
Input Channel Size: ICH
Output Channel Size: OCH
To ⇐ 1
Ti ⇐ 1
while To × Ti × 2 ≤ DSP_NUM × 2 do
  if Ti < To  &&  Ti × 2 ≤ ICH then
    Ti  × = 2;
  else if To ×2 ≤ OCH then
    To  × = 2;
  else
    break;
  end if
end while
```

In addition, the weight required for the calculation must be fetched from the DDR4 SDRAM. In this implementation, double buffering is performed to hide the time required to fetch the weights, two buffers that hold the weights are prepared, and they are used alternately for the calculation. As a result, while the calculation is performed using the data in one buffer, the weights required for the next calculation can be fetched using the other buffer. The processing time is not the sum of the time for the calculation and the fetching of the weights; rather, it is equivalent to the processing time of only the one that takes more time than the other. From the policies described so far, the algorithm for the convolution operation is shown in Algorithm 2. Double buffering, pipeline processing, and loop unrolling are performed by inserting the pragma provided by Vivado HLS. As shown in Fig. 10, at the same time as fetching the weight with one Weight Buf, the other Weight Buf and the Input Feature Map are used to execute the multiplication of $To \times Ti$ in parallel. The result is accumulated in the Output Buf. In the example of this figure, the case of $To = 2$ and $Ti = 3$ is shown, and six multiply and accumulate operations are performed per clock. The weight data are pre-sorted in the order of use, and they can be efficiently fetched by burst transfer. Because all operations on the fetched weights are completed before the operation on the next weights is completed, all the weights are completely reused and the number of DRAM accesses is minimized. After these processes, the intermediate result is subjected to bias addition and scaling processes and is held in the on-chip buffer as the Output Feature Map. This output feature map is used as an input feature map of the next layer.

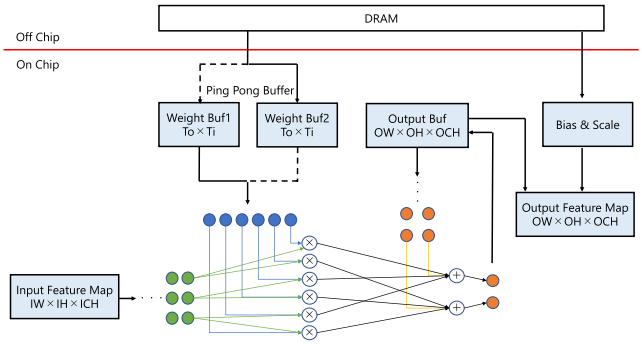


Fig. 10 Architecture of CONV Layer.

Algorithm 2 Convolution operation

```
Require: input_feature_map:IW × IH × ICH;  output_buf:OW × OH ×
  OCH;  weight_buf:To × Ti
Ensure: output
  for to = 0; to < OCH/To; to + + do
    for ti = 0; ti < ICH/Ti; ti + + do
      for each(ky, kx) within(K, K) do
        pragma HLS DATAFLOW
        load_weight(weight_buf, ddr)
        for each(h, w) within(OH, OW) do
          pragma HLS PIPELINE
          for too = 0; too < To; too + + do
            pragma HLS UNROLL
            for tii = 0; tii < Ti; tii + + do
              pragma HLS UNROLL
              output_buf[w][h][to×To+too]+= input_feature_map
              [S × w + kx − P][S × h + ky − P][ti × Ti + tii] ×
              weight_buf[too × Ti + tii]
            end for
          end for
        end for
      end for
    end for
  end for
```

6. Evaluation

6.1 Experimental Result

In this implementation, an IP of the calculation part is generated from the code written in C++ with the high-level synthesis tool Vivado HLS 2019.1.3 and implemented with the synthesis tool Vivado 2019.1.3. A rather conservative operational frequency of 100 MHz is adopted for easy design. The resource utilization of each board in the implementation on 4 boards after placement and routing is shown in Fig. 11. It is evident that this implementation can utilize a considerable amount of the on-chip memory and DSPs.

The execution time and power consumption of each board are listed in Table 4. The power consumption was evaluated with a real board of the cluster. Analog Device DC2086A and DC1613A were used to monitor the supply current and voltage. AMD LTpowerPlay was used for control software. All power for the system, FPGA core, I/O, high-speed links, and DDR SDRAM modules are included. The data of power consumption are an average of 5 seconds during execution. Since the performance is determined by the part with the longest execution time in pipeline processing, it is possible to process one data every 13.3 ms in this implementation. Of this time amount, the time required for communication between boards is approximately 0.6 ms at most; thus, there is no significant impact on overall performance. In the latter layer of ResNet, communication with DRAM is bound owing to the size of the number of weights, and the execution time per layer is long. The execution time of board 3 is shorter than that of other boards. However, if more layers are assigned to board 3, the execution time of board 3 exceeds 13.3 ms, so allocation of our work is optimal. As a result, the throughput is $1/0.0133 = 75.1$ FPS. It can be said that it has sufficient ability to process 60 FPS videos, which is one of the general frame rates. In addition, because the total calculation amount of ResNet-50 is 3.88×10^9, the performance is $3.88 \times 10^9/0.0133 = 292$ GOPS. The power efficiency is obtained from the performance and total power consumption, and this implementation achieves 7.81 GOPS/W.

Figure 12 shows the performance when ResNet-50 was implemented on 2, 3, 4, 5, and 6 boards. Implementation on a single board was not possible due to the unsuccessful placement and routing on Vivado. This is one of the reasons why huge CNN models should be implemented on multiple FPGAs. From 2 boards to 5 boards, it can be observed that the performance improves linearly with the increase in the number of M-KUBOS boards. However, from 5 to 6 boards, the improvement is slower. In the convolution layer, where the number of parameters is particularly large, the communication time with DRAM becomes a bottleneck and the execution time becomes large. This makes the finer grain partitioning difficult and prevents the linear speedup.

From Table 2, it can be seen that the number of parameters is larger for the residual block at the back. In this implementation, conv2_x, conv3_x, and conv4_x are compute-bound, while conv5_x is memory-bound. The execution time required for conv2_x, conv3_x, and conv4_x is approximately 150,000 clocks per residual block, on the other hand, each residual block of conv5_x requires about 400,000 clocks. Since the split in this implementation is performed in residual block units, conv5_x interferes with distributing layers such that each board's execution time is even. This is the main reason why the performance improvement with 6 boards is saturated. If the number of boards becomes large, balancing the load on each board tends to be difficult. Thus, although we can expect a small speed up by distributing the load of the heaviest board, increasing the number of boards by more than five is not desirable from the viewpoint of energy efficiency.

In this implementation, the performance of the serial link did not become a performance bottleneck. Even if the link performance was not as high as the M-KUBOS cluster, a similar performance was likely obtained.

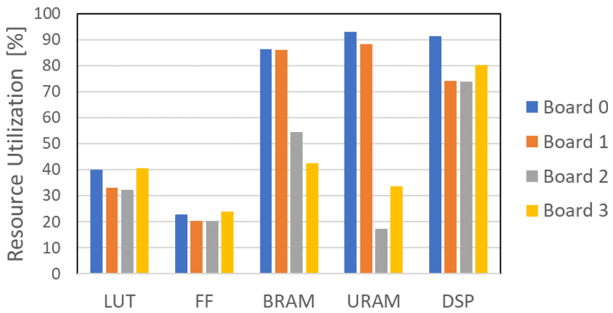


Fig. 11 Resource utilization of each board.

Table 4 Evaluation of each board.

	Board 0	Board 1	Board 2	Board 3
Execution Time (msec)	12.7	13.3	11.6	9.7
Power Consumption (W)	9.49	9.40	9.11	9.37

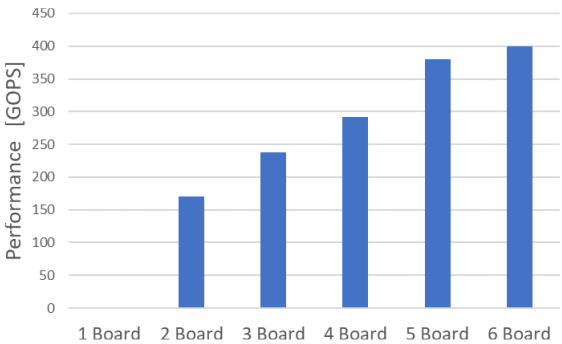


Fig. 12 Performance when implementing on 2-6 boards.

6.2 Performance Comparison

The execution time, power consumption, and power efficiency of this implementation on 4 boards are compared with those of the CPU and GPU, as shown in Table 5. For comparison, the AMD Ryzen Threadripper 3990X was used

Table 5 Performance comparison with CPU and GPU.

	CPU	GPU	Embeded GPU	Our Implementation
Device	AMD Ryzen Threadripper 3990X	NVIDIA GeForce RTX 3090	NVIDIA Jetson Nano	Xilinx Zynq UltraScale+
Frequency (MHz)	2900	1400	N/A	100
Software implementation	PyTorch 1.8.1 + Python3.6.8	PyTorch 1.8.1 + Python3.6.8 + CUDA 11.1	Hello AI World [20]	Xilinx Vivado HLS 2019.1.3
Precision	Float32	Float32	Float32	INT8
Top-1 Accuracy (%)	74.4	74.4	-	74.2
Top-5 Accuracy (%)	93.0	93.0	-	92.8
Latency (msec)	230.7	8.2	166.5	13.3
Performance (GOPS)	17	473	23.7	292
Power Efficiency(GOPS /W)	0.06	1.35	2.37	7.81

Table 6 Performance comparison with the related work and Vitis AI implementation.

	Our Implementation (4-board)	Our Implementation (5-board)	[15] (4-board)	[15] (16-board)	Vitis AI Implementation (single board)
Device	Xilinx Zynq UltraScale+ (XCZU19EG)	Xilinx Zynq UltraScale+ (XCZU19EG)	Xilinx Virtex UltraScale (XCVU095)	Xilinx Virtex UltraScale (XCVU095)	Xilinx Zynq UltraScale+ (XCZU9EG)
Frequency (MHz)	100	100	150	150	281
CNN Model	ResNet-50	ResNet-50	ResNet-152	ResNet-152	ResNet-50
Precision	INT8	INT8	Fixed16	Fixed16	INT8
Performance (GOPS)	292	380	62.9	256.6	338.9

as the CPU, and NVIDIA GeForce RTX 3090 was used as the GPU. The deep learning framework PyTorch was used for both the CPU and GPU, and the inference was executed and evaluated using the pre-trained ResNet-50 model. The CPU ran with 64 threads, which is the default thread number of PyTorch. It also showed a comparison with Nvidia Jetson Nano, a highly efficient embedded GPU. We ran ResNet-50 inference using Hello AI World [20] and evaluated the execution time. Jetson Nano was operated in MAXN mode, which uses up to 10 W.

As shown in Table 5, this implementation on 4 boards achieved 17 times faster speed and 130 times more power efficiency than the CPU deep learning framework implementation. Although it was inferior to the GPU in terms of speed, it achieved a 5.8 times improvement in terms of power efficiency. Note that the GPU used here must be connected to a host server, and such a system is difficult to introduce to a base station as a MEC server, while the PYNQ cluster consists of four small boards with only a simple power supply and a fan. In addition, our implementation shows competitive results even compared to embedded GPUs.

A comparison with the related research [15] that implements the CNN in the multi-FPGA system is shown in Table 6. As in this research, ResNet was implemented on multiple FPGA boards. Thus, it can be stated that it is appropriate as a comparison target. ResNet has the same basic structure even if the number of layers is different. By using GOPS instead of FPS as an index of performance, the difference in the total amount of calculation can be almost ignored. It is therefore possible to compare these models. Compared to the 4-board implementation of [15], this implementation achieved 4.6 times its performance. In [15], the utilization rate of DSP was only approximately 20%. It can be inferred that the DSP could not be allocated efficiently in the division. On the other hand, in our work, by using a common design for the convolution layer, the DSP utilization rate was high and the performance was higher than that of [15]. In addition, a comparison with the implementation using Vitis AI is also shown in Table 6. Since Vitis AI does not work on the M-KUBOS board, we have shown a comparison with the ZCU102, which is close to what we used in our implementation. We refered to the performance published by Xilinx in [21]. Our 4-board implementation is inferior in performance to the implementation using Vitis AI, but our 5-board implementation is superior to the implementation using Vitis AI. Such scalability is the strength of multi-FPGA implementations.

Note that the frequency of FPGA and communication between FPGAs are not specialized for this implementation because it is assumed that the implementation of communication and frequency in the FPGA cluster is used as the server of MEC. In addition, because this implementation focuses on mounting on multiple boards and does not incorporate speed-up methods such as the Winograd algorithm [22], FFT [23], and pruning [24], there remains room for further performance improvement. Also, to avoid performance saturation as the number of boards increases, it is conceivable to prepare implementations with a small number of boards and use several of them independently. It is our future work.

7. Conclusion

In our work, as an example of image recognition for MEC applications, ResNet-50 was divided and implemented on a PYNQ cluster consisted of M-KUBOS boards. The throughput was improved by performing pipeline processing between the multiple boards. The speed was further in-

creased by using quantization to an 8-bit integer, loop unrolling, and double buffering. By using our PYNQ cluster in which FPGAs were directly connected by high-speed serial links, stream processing without network bottlenecks and pipeline processing between boards could be realized easily. The proposed implementation on 4 boards achieved a throughput of 75.1 FPS, performance of 292 GOPS, and power efficiency of 7.81 GOPS/W. It achieved 17 times faster speed and 130 times greater power efficiency compared to the implementation on the CPU and 5.8 times greater power efficiency compared to the implementation on the GPU. Also, it can be observed that the performance improves linearly with the increase in the number of M-KUBOS boards from 2 boards to 5 boards.

There remains room for performance improvement, such as by using other CNN speed-up methods. Because ResNet has multiple models with a maximum of 152 layers, it is possible to expand it step by step in the future. Further research is needed on the division method, and automation of division is a future challenge.

Acknowledgments

This work was supported by JST CREST, Grant Number JPMJCR19K1, Japan.

References

[1] PALTEK, "FPGA computing platform M-KUBOS," https://www.paltek.co.jp/design/original/m-kubos/ (accessed 2021-1-20).
[2] T. Inage, K. Hironaka, K. Iizuka, K. Ito, Y. Fukushima, M. Namiki, and H. Amano, "M-KUBOS/PYNQ cluster for multi-access edge computing," CANDAR2021, Nov. 2021.
[3] Xilinx Inc, "PYNQ - Python productivity for Zynq - Home," http://www.pynq.io/ (accessed 2021-1-22), 2019.
[4] K. Ito, K. Iizuka, K. Hironaka, Y. Hu, M. Koibuchi, and H. Amano, "Implementing a multi-ejection switch and making the use of multiple Lanes in a circuit-switched multi-FPGA system," CANDAR20 Workshop, Nov. 2020.
[5] K. Hironaka, K. Iizuka, M. Yamakura, A.B. Ahmed, and H. Amano, "Remote dynamic reconfiguration of a multi-FPGA system FiC (Flow-in-Cloud)," IEICE Trans. Inf. & Syst., vol.E104-D, no.8, pp.1321–1331, Aug. 2021.
[6] "Slurm Workload Manager." https://slurm.schedmd.com/ (accessed 2021-12-01).
[7] M. Yamakura, R. Takano, A.B. Ahmed, M. Sugaya, and H. Amano, "A multi-tenant resource management system for multi-FPGA systems," IEICE Trans. Inf. & Syst., vol.E104-D, no.12, pp.2078–2088, Dec. 2021.
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016 IEEE Conference on Comput. Vis. Pattern Recognit. (CVPR), pp.770–778, June 2016.
[9] MLCommons, "MLPerf," https://mlcommons.org/ja/ (accessed 2021-6-18).
[10] Microsoft Research, "Project Brainwave," https://www.microsoft.com/en-us/research/project/project-brainwave/ (accessed 2021-6-18).
[11] H. Sharma, J. Park, D. Mahajan, E. Amaro, J.K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.1–12, 2016.
[12] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," Proc. 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '15, New York, NY, USA, pp.161–170, ACM, Feb. 2015.
[13] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W. Hwu, and D. Chen, "DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs," 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp.1–8, Nov. 2018.
[14] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," Proc. 2016 International Symposium on Low Power Electronics and Design, ISLPED '16, New York, NY, USA, pp.326–331, Association for Computing Machinery, Aug. 2016.
[15] W. Zhang, J. Zhang, M. Shen, G. Luo, and N. Xiao, "An efficient mapping approach to large-scale DNNs on multi-FPGA architectures," 2019 Design, Automation Test in Europe Conference Exhibition (DATE), pp.1241–1244, March 2019.
[16] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, "A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp.394–398, 2018.
[17] Y. Fukushima, K. Iizuka, and H. Amano, "Parallel implementation of CNN on multi-FPGA cluster," MCSoC-2021, Dec. 2021.
[18] PyTorch, "Quantization — PyTorch 1.9.0 documentation," https://pytorch.org/docs/stable/quantization.html (accessed 2021-6-18).
[19] D. Nguyen, D. Kim, and J. Lee, "Double MAC: Doubling the performance of convolutional neural networks on modern FPGAs," Design, Automation & Test in Europe Conference Exhibition (DATE), 2017, pp.890–893, 2017.
[20] NVIDIA Developer, "Two Days to a Demo," https://developer.nvidia.com/embedded/twodaystoademo (accessed 2022-12-26).
[21] Xilinx, "AI-Model-Zoo," https://github.com/Xilinx/Vitis-AI/tree/master/model_zoo (accessed 2022-12-26).
[22] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.39, no.4, pp.857–870, April 2020.
[23] C. Zhuge, X. Liu, X. Zhang, S. Gummadi, J. Xiong, and D. Chen, "Face recognition with hybrid efficient convolution algorithms on FPGAs," Proc. 2018 on Great Lakes Symposium on VLSI, GLSVLSI '18, pp.123–128, New York, NY, USA, Association for Computing Machinery, May 2018.
[24] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.17–25, 2019.

Yasuyu Fukushima received B.E. degree from Keio University, Yokoghama, Japan, in 2021. He is a master student in the Department of Information and Computer Science, Keio University in the presence. His reserch interests are deep learning accelerators and reconfigurable architectures.

Kensuke Iizuka received B.E. degree from Keio University, Yokoghama, Japan, in 2018 and M.E. degree from Keio University, Japan, in 2020. He is a Ph.D. student in the Department of Information and Computer Science, Keio University in the presence. His reserch interests are deep learning accelerators and reconfigurable architectures.

Hideharu Amano received Ph.D. degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.