Automated FPGA Implementation of Convolutional Neural Networks with Pipelining and Layer Partitioning

Eito YAMADA Kazuyoshi TAKAGI
Department of Information Engineering, Graduate School of Engineering, Mie University
Tsu city, Mie 514-8507, Japan
{yamada,ktakagi}@arch.info.mie-u.ac.jp

Abstract— Convolutional neural networks (CNNs) are used in various machine learning applications. In this work, we show an acceleration scheme for CNN processing using field programmable gate arrays (FPGAs). High throughput operation is achieved by pipelining operation and layer partitioning. We also propose an automated design flow to map CNN operations on FPGA. In our experiments, the CNN operations for the Fashion-MNIST and CIFAR-10 datasets are about 140 to 250 times faster compared to CPU execution.

I. Introduction

Convolutional neural networks (CNNs) have a wide range of machine learning applications, e.g., in the field of image processing such as industrial robotics and automated driving technology. The operation speed is crucial in CNN inference more than in learning, because real-time performance is often required. With the demand for real-time data processing, field programmable gate arrays (FPGAs) are attracting attention for the high-speed and low-energy processing potential. Thus, studies have been made on accelerating CNN operations using FPGAs. It is a common strategy to exploit parallelism, pipelining and hardware-oriented quantization in the accelerator design[1, 2]. High-level synthesis (HLS) tools are often used in the implementation.

In this paper, we propose optimization methods of CNN data flow on FPGA by adjusting the amount of operations in a stage with parallelization and layer partitioning. Based on the optimization methods, we also propose an automated design flow. The advantage of the flow is that it integrates pipelining, parallelization, and layer partitioning considering the nature of CNNs.

II. Convolutional Neural Network

CNN is a kind of neural networks with a structure in which features are extracted repeatedly in the convolutional and pooling layers, and then passed through the fully connected layer. A large portion of the CNN computation time is spent in the convolution layer. In the convolution operation shown in Fig. 1, there is no dependence on the operations for different i, j, and c, where $i \times j$ is the size of the input feature map and c is the number of output feature maps. Therefore, the operations can be processed in parallel.

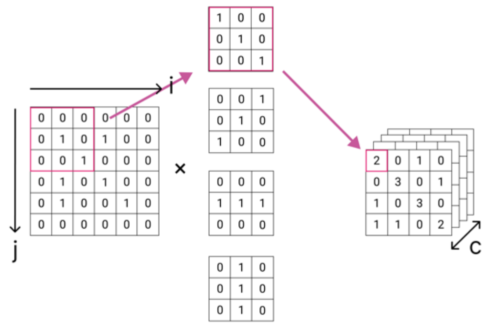

Fig. 1. Convolutional operation

III. Acceleration of CNN Processing

A. CNN Operations on FPGA

We assume a CNN processing system consisting of a CPU for input/output data management and an FPGA for computation of the whole CNN layers. The input data are converted into appropriate data format and sent to the FPGA. We employ common technique of parallelization and pipelining used in FPGA accelerators for CNNs, and further optimize the dataflow by layer partitioning. Parameters such as weights and biases are stored in the FPGA memory in advance. After the computation of the CNN layers is done, the results are sent back to the CPU.

B. Dataflow Optimization

We employ dataflow optimization methods as shown in the following three steps. In the experiments, these steps are adopted in sequence. That is, cases with step 1, steps 1 plus 2, and steps 1 plus 2 plus 3 are examined.

Step 1 Pipeline processing of each layer
Step 2 Parallelization of time-consuming layers
Step 3 Partitioning of time-consuming layers

The baseline dataflow of step 1 is layer-by-layer pipeline processing for multi-layered CNNs. This structure enables higher throughput while maintaining low hardware resource usage.

Now, the overall processing time of the CNN depends on the slowest processing layer which decides the cycle time per pipeline stage. With step 2, additional operation units are assigned to several slowest layers and the operations are processed in parallel through multiple pipelines.

If further speedup is required after repeated application of step 2, we partition several slowest layers to be processed in multiple stages of the pipeline, in step 3. There can be a trade-off between cycle time and the number of stages.

C. *Design Automation*

Finding optimal parameters in steps 2 and 3 requires repeated synthesis and optimization. We implemented an automated design tool for this procedure as follows. Here, we exploit the nature of the typical network that a few layers are extremely large and others are relatively small.

1. Construct each layer as a pipeline stage.
2. Classify the layers into large layers (class $\mathcal{L}$ to be transformed) and small layers (class $\mathcal{S}$ not to be transformed), and let c_{max} be the maximum number of cycles among class $\mathcal{S}$.
3. For each layer l in class $\mathcal{L}$, in descending order of the computational cost, parallelize layer l to be fit within c_{max} cycles using additional operation units. If layer l does not fit within c_{max} cycles, replace c_{max} with the minimum number of cycles for layer l.
4. Identify layers with the largest and the second largest number of cycles, layers l_a and l_b with c_a and c_b cycles respectively.
5. If $c_a/c_b \geq r$ (a constant), partition l_a into $2, 3, 4, \ldots$ pipeline stages and select the best configuration.
6. Synthesize the whole network calculation.

The constant r in line 5 is set to 1.4 in our experiments. Though we performed the partition step (lines 4 and 5) only once, it can be repeated several times.

IV. Experiments

A. *Environment and Datasets*

The source code for CNN computation is written in C++ and synthesized using Vitis HLS. The design target is Xilinx Alveo U50 FPGA accelerator. Execution on Intel Xeon Silver 4214 CPU is also inspected for comparison.

We used two types of image processing datasets. CIFAR-10 is an image dataset of color photos of 10 kinds of vehicles and animals. Fashion-MNIST is an image data set of 10 fashion product photos. The layer structure of the CNN used is as follows.

- CIFAR-10: 4 convolution layers, 4 pooling layers, 2 fully-connected layers, 5 activation function layers.
- Fashion-MNIST: 2 convolution layers, 2 pooling layers, 2 fully-connected layers, 3 activation function layers.

TABLE I
Performance of Image Processing (frames/second)

system	Fashion-MNIST	CIFAR-10
CPU	165.59	26.56
FPGA (baseline)	73.84	8.86
FPGA (Step 1)	79.32	14.44
FPGA (Steps 1+2)	8680.56	3324.47
FPGA (Steps 1+2+3)	22988.51	6644.52

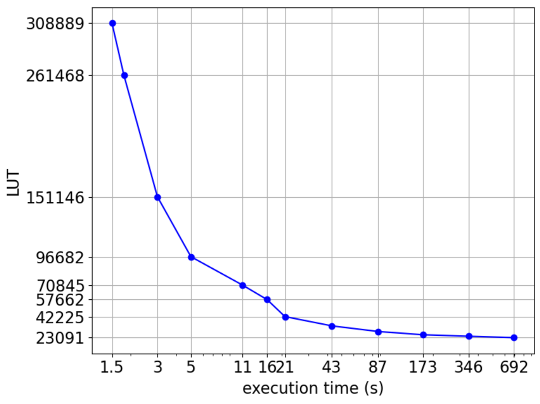

Fig. 2. Execution time vs. LUT usage for CIFAR-10 processing

B. *Evaluation*

Table I shows the execution time with each parallelization method. The baseline is a result of high-level synthesis of the same source code as the CPU, and the parallelization steps are applied in sequence.

The FPGA configurations are obtained using the automated design flow. The execution time and the LUT resource usage for CIFAR-10 processing, with a range of execution time constraints, is plotted in Fig. 2. The automated design tool could quickly obtain the same results as the manually optimized configuration for these datasets.

V. Conclusion

The proposed parallelization methods take advantage of FPGAs to minimize the resource usage by performing pipeline processing, parallelization, and layer partitioning with appropriate parameters at each layer. We consider the automated design flow is generic and effective also for larger networks, because it can quickly explore the whole design space supposed in manual design. The evaluation is left for our future work.

References

[1] C. Huang, S. Ni, and G. Chen, "A layer-based structured design of CNN on FPGA," *IEEE 12th International Conference on ASIC (ASICON), 2017.*

[2] S. G. Aydin, H. S. Bilge, "FPGA-based implementation of convolutional layer accelerator part for CNN," *Innovations in Intelligent Systems and Applications Conference (ASYU), 2021.*