ICIC Express Letters
Volume **10**, Number **7**, July **2016**

 ISSN 1881-803X
pp. **1535–1540**

A LOOP-BASED PARTITION SCHEME FOR CONVOLUTION FILTERING ON FIELD PROGRAMMABLE GATE ARRAYS

Zhijian Lu

Postdoctoral Scientific Research Workstation
Shanghai Futures Exchange
No. 500, Pudian Road, Shanghai 200122, P. R. China
lu.zhijian@shfe.com.cn

Received November 2015; accepted February 2016

Abstract. *Convolutional networks are computational models that are widely used in pattern recognition applications. Operations in these applications require convolution filtering at each pixel. Depending on the size of input image and convolution kernel, the two-dimensional convolution can require significant amounts of computation, thus suggesting a highly parallel implementation in hardware. This paper describes a loop-based computation and storage partition scheme for convolution filtering on Field Programmable Gate Arrays (FPGAs), which can guide the proper parameterization of a convolution filter bank on a given architecture. The case study on selected kernels shows the method can express the computation and storage tradeoffs involved in mapping a Convolutional Neural Networks (CNNs) on FPGAs.*
Keywords: Convolutional neural networks (CNNs), Loop-based partition, Field programmable gate arrays (FPGAs)

1. **Introduction.** Multi-layer Convolutional Neural Networks (CNNs) are feed-forward multi-layer architectures composed of multiple layers of neurons [1,2]. Every connection between an input neuron and an output neuron is assigned a value called connection weight. Each output neuron computes a weighted multiply-add sum of all its inputs, followed by a non-linearity and a feature pooling stage. The input and output of each layer are sets of arrays called feature maps. At the output, each feature map represents a particular feature extracted from the inputs. In CNNs, each layer is composed of three stages: a filter bank stage, a non-linearity and a feature pooling stage [3]. The complete CNN architecture is composed of less than three such three-stage layers, followed by a fully-connected classifier. Consider an example in real time video processing, the average pixel rate required is 9.2 Mpixels/sec at least for a VGA (640×480 pixels) resolution. In the case of a high definition application with 1280×720 resolution, the required pixel rate is approximitely 20 Mpixels/sec. When it comes to full HD (1920×1080 pixels) resolution, the pixels rate is more than 62 Mpixels/sec. For typical tasks, the number of operations for each pixel is 10 to 100 or more, which requires vast computation power. Obviously only using highly parallel computing structures can meet these requirements.

Due to the loop-based computation pattern, general purpose processors can not meet the performance requirement. Various schemes based on FPGA or GPU have been proposed recently to improve performance of CNN designs [4-6]. Some efforts have implemented convolution operations in hardware [7,8], and others like large-category image classification and automatic speech recognition require substantial computing resources to train and evaluate [6,9]. The fast convolution implementation itself is not sufficient to accelerate the entire convolution network. Some optimizations are required to hold weight values and intermediate data for the acceleration. For a given number of processing elements and external memory bandwidth, explorations of architectural configuration are

needed to find the architecture parameters for the convolution networks, that can match the multiple parallelism and give the best throughput performance.

This paper presents a loop-based computation and storage partition scheme for convolution filtering on field programmable gate arrays. For a given number of computation units and memory bandwidth, the scheme can find the proper architecture parameters by exploring the architecture configurations, which can exploit the massive parallelism in hardware and give the best throughput. The rest of the paper is organized as follows. Section 2 presents a loop-based convolution filtering model and partition scheme, which can express the storage and computation trade-offs involved in mapping CNNs on FPGAs. Section 3 shows the cost functions with different hardware constraints and two practical scenarios for a case study. And Section 4 concludes the paper.

2. Computation Partition of Convolution Filtering Array.

This paper will focus on the filter array sub-stage of the feed-forward propagations. However, the back-propagation phase of convolution networks that trains the convolution kernel values is not considered. Feed-forward propagations are composed of multiple layers. The input and output of each layer are sets of feature maps. Each layer is composed of three sub-stages: a convolution filter array, a non-linearity, and a feature pooling sub-stage. A typical CNN is composed of less than three such layers, followed by a classifier. In this section, different mapping strategies are used to show that the scheme could correctly capture the various architecture scenarios, which has different execution times for a given FPGA architecture.

2.1. Loop-based convolution filtering model.

The descriptions below focus on computing layer $l+1$ based on layer l of feed-forward propagation, and the outputs are the feature maps of layer $l+1$. The input is a three-dimensional array with F_l two-dimensional feature maps of size $X_l \times Y_l$. Each element is denoted n_{ijk} and each feature map is denoted f_l. The output is also a three-dimensional array with F_{l+1} feature maps of size $X_{l+1} \times Y_{l+1}$. A trainable convolution kernel k_{ij} in the sublayer has size $K \times K$ and connects input feature map f_l to output feature map f_{l+1}. A filter array sub-stage accepts I feature maps f_i from layer l as inputs and produces J intermediate outputs I_j. To produce the intermediate output I_j, the input feature maps are first individually convolved with convolution kernels k_{ij} and the convolved results are accumulated. Then a trainable bias value is added to the accumulated result. However, the dimension of intermediate output array can vary from one convolution network to another. The loop-based representation of convolution filtering model is shown in Algorithm 1. From the computational point of view, $N_i \times N_j$ convolution kernel computations are the most computationally intensive parts of the convolution networks, especially when the kernel size and input image are large. X_l is the feature map width in layer l. ΔX is the moving step length in row direction. Y_l is the feature maps length in layer l. ΔY is the moving step in column direction. p^l is the output value in layer l.

2.2. Splitting computations.

The abundant parallelism is inherent to CNN computation models. For the hardware constraints, it is not possible to have all feature maps and the intermediate values fit in the on-chip storage, nor to have all kernels fit on the computational units. Trade-offs are required to split the computations. In order to formally express and show that, each loop is splited into two loops. The original loop l is split into blocks of size ll, resulting in two loops. Potentially, all computations can be splitted along those principles, as shown in Algorithm 2. Therefore, the main contribution of this representation is that it becomes fairly simple to express certain computations, in order to adapt to on-chip storage, or to the number of available operators.

Algorithm 1: Loop-based representation of convolution

for $f_{l+1} = 1$ *to* $f_{l+1} = N_j$ **do**
  for $f_l = 1$ *to* $f_l = N_i$ **do**
    for $x_l = 1$ *to* $x_l = X_l$ *by* ΔX **do**
      $x_{l+1} = \frac{x_l}{\Delta X}$
      for $y_l = 0$ *to* $y_l = Y_l - 1$ *by* ΔY **do**
        $y_{l+1} = \frac{y_l}{\Delta Y}$
        initialize v
        for $i = 1$ *to* $i = N_i$ **do**
          for $j = 1$ *to* $j = N_j$ **do**
            $v+ = p^l_{f_{(i+x_l),(j+y_l)}} \times w^{l,l+1}_{ij}$
        $p^{l+1}_{f_{x_{l+1},y_{l+1}}} = v$

Algorithm 2: Splitting the computations in loop-based representation

for $ff_{l+1} = 1$ *to* $ff_{l+1} = F_l$ *by* FF_{l+1} **do**
  for $f_l = ff_l$ *to* $f_l = ff_l + FF_l$ **do**
    for $ff_l = 1$ *to* $ff_l = F_l$ *by* FF_l **do**
      for $f_{l+1} = ff_{l+1}$ *to* $f_{l+1} = ff_{l+1} + FF_l$ **do**
        for $xx_{f_l} = 1$ *to* $xx_{f_l} = X_{f_l}$ *by* XX_{f_l} **do**
          for $x_{f_l} = xx_{f_l}$ *to* $x_{f_l} = xx_{f_l} + XX_{f_l}$ **do**
            $x_{f_{l+1}} = \frac{x_{f_l}}{\Delta X^{f_{l+1}}_{f_l}}$
            for $yy_{f_l} = 1$ *to* $yy_{f_l} = Y_{f_l}$ *by* YY_{f_l} **do**
              for $y_{f_l} = yy_{f_l}$ *to* $y_{f_l} = yy_{f_l} + YY_{f_l}$ **do**
                $y_{f_{l+1}} = \frac{y_{f_l}}{\Delta Y^{f_{l+1}}_{f_l}}$
                initialize v
                for $i = 1$ *to* $i = N_i$ **do**
                  for $j = 1$ *to* $j = N_j$ **do**
                    $v+ = p_{f_l}(i + x_{f_l}, j + y_{f_l}) \times w^{f_l,f_{l+1}}_{ij}$
                $p_{f_{l+1}}(x_{f_{l+1}}, y_{f_{l+1}}) = v$

2.3. **Mapping scheme of loop partition.** It is possible that there are not enough operators for implementing all the kernels required for all output feature maps. In that case, it is possible to apply this reuse the input feature map data for only a fraction of the output feature maps, thus requiring to implement fewer kernels. This can be expressed again by moving loop ff_l above the loops which have been previously moved. This also has the effect of reducing the storage requirements for the kernels weights. However, since all the output feature maps can no longer be computed in one time, the data of the input feature maps have to be reloaded several times, as many times as the number of blocks of output feature maps, i.e., $\frac{F_l}{FF_l}$.

2.4. **Dependence analysis.** While several loops have been interchanged, thereby expressing the reuse of data, or the reduction of computational requirements, not all transformations are possible because of dependences. Consider for instance loop f_l and loops

xx_{f_l}, yy_{f_l}. Because the latter indices depend on f_l, they cannot be moved above this loop. This corresponds to the case where each input feature map has a different dimension, hence the index is f_l. Since moving xx_{f_l}, yy_{f_l} above f_l means that the *same* portion of all feature maps is meant to be reused, the notion is potentially valid since all feature maps do not have the same number of blocks.

3. **Mapping Computations to Convolution Kernel Primitives.** The goal of using different mapping strategies is to show that the model could correctly capture the various scenarios, which has different execution times for a given FPGA architecture.

3.1. **Convolution computation with loop unrollling.** The implementation of 2D convolution is not trivial because it is not only compute-extensive but also memory-intensive. With $N \times N$ convolution kernel, it requires $N \times N$ multiplications and $N \times N - 1$ additions, as well as $N \times N$ accesses to the input image data for the calculation of a single output pixel.

The convolution computation includes a post accumulation to allow the combination of multiple convolutions. It shows an example pipeline implementing a $N \times N$ convolver. Each pixel is loaded line by line and is multiplied by all N^2 connection weights. Partial sums are then accumulated by the adder, with a delay of one pixel between adjacent adds and a delay of the input image row length minus N pixels between rows for proper alignment. This also has the effect of reducing memory bandwidth requirements. When a pixe value was loaded into the on-chip buffers, it could be reused for calculation of successive windows to avoid fetching it repetitively from external memories. As a result, the requirement for external memory bandwidth was reduced to acceptable level. The time for completing one input feature map is simply the time required to fetch the image from memory one pixel at a time.

Iterations of a loop are continuously initiated at constant intervals, without having to wait for preceding interations to complete, which is shown in Algorithm 3. The advantage of loop unrolling is that optimal performance can be achieved with minimum memory bandwidth. The loop unrolling scheme in this paper is a practical, efficient technique for scheduling the parallelism within a convolution operation. The convolution primitives exploit the parallelism inherent in a convolution using a streaming architecture.

Although there are various ways to implement 2D convolution, note that this pipeline is optimal in the case that each input pixel is accessed only once. Output values are produced at the same rate, so maximum parallelism is archieved for the given memory bandwidth. Values from the input plane are put into N on-chip FIFOs of which the size is the width of the image minus the width of the kernel. Shifting values in these FIFOs correspond to shifting the convolution window over input plane. At each clock cycle, values are shifted by one, and the inner product between the input plane window and the kernel is computed in parallel.

3.2. **Constraints analysis.** The constraints on FF_l and FF_{l+1} brought by the FPGAs are the following. The product of FF_l and FF_{l+1} must be less than or equal to the total hardware available which means $FF_l \times FF_{l+1} \leq T_k$, in which FF_l and FF_{l+1} are the number of input feature maps and output feature maps respectively, T_k is the total number of kernels that can simultaneously fit on the FPGA.

Other constraints are brought by the memory bandwidth. If FF_l is used in reading inputs in layer l, where N_b is the number of bits used for each point within a layer, T_b is the total number of bits per cycle for each memory port transferring, then $FF_l \times N_b \leq T_b$. If FF_l is used in reading intermediate data, where I_b is the intermediate data bit wide, then $FF_l \times I_b \leq T_b$. If FF_{l+1} is used in writing intermediate data, then $FF_{l+1} \times I_b \leq T_b$. If FF_{l+1} is used in writing final outputs, then $FF_{l+1} \times N_b \leq T_b$.

Algorithm 3: Convolution computation with loop unrolling

for $f_{l+1} = 1$ *to* $f_{l+1} = F_{l+1}$ **do**
  for $f_l = 1$ *to* $f_l = F_l$ **do**
    for $x_{f_l} = \Delta X_{f_l}^{f_{l+1}}$ *to* $x_{f_l} = \Delta X_{f_l}^{f_{l+1}} + X_{f_l}$ *by* $\Delta X_{f_l}^{f_{l+1}}$ **do**
      $x_{f_{l+1}} = \frac{x_{f_l}}{\Delta X_{f_l}^{f_{l+1}}}$
      for $y_{f_l} = \Delta Y_{f_l}^{f_{l+1}}$ *to* $y_{f_l} = \Delta Y_{f_l}^{f_{l+1}} + Y_{f_l}$ *by* $\Delta Y_{f_l}^{f_{l+1}}$ **do**
        $y_{f_{l+1}} = \frac{y_{f_l}}{\Delta Y_{f_l}^{f_l+1}}$
        for $i = 1$ *to* $i = N_i$ **do**
          for $j = 1$ *to* $j = N_j$ **do**
            $reg_{i,j+1} <= reg_{i,j} + p_{f_l}(x_{f_l}, y_{f_l}) \times w_{ij}^{f_l, f_{l+1}}$
            if $(i \neq N-1)$
            for $d = N$ *to* $d = (X-1)-1$ **do**
              $reg_{i,d+1} <= reg_{i,d}$
              $reg_{i+1,0} <= reg_{i,X-1}$
              $reg_{output} <= reg_{i,N}$
            $p_{f_{l+1}}(x_{f_{l+1}}, y_{f_{l+1}}) = reg_{output}$

When computing the convolution filter layer without reading the temporary intermediate, the number of memory bits read per cycle is $FF_{l-1} \times N_b$. If there exist temporary results, then $FF_l \times N_b + FF_{l+1} \times I_b$ bits are read per cycle. The amount of memory bits written per cycle is $FF_{l+1} \times I_b$ or $FF_{l+1} \times N_b$ depending on whether intermediate or final outputs are kept. T_b bits per cycle is the upper limit of read requests or write requests, which cannot be exceed.

3.3. **Case study.**

3.3.1. *Independent output within single pass.* FF_l convolvers simultaneously process FF_l different input feature maps, and FF_l convolutions are combined to yield a single output. With processing FF_l different input feature maps, every outputs are computed in a serial pattern. However, each output is being computed in parallel, that is FF_l feature maps are being precessed in parallel. Note that there is no intermediate data to write out since no partial outputs are computed. The execution time is roughly the same as it takes to read one input feature map about F_{l+1} times, which can be expressed by moving the x and y loops above the f_l loops, which can be expressed in Algorithm 4.

The number of passes required for completion is F_{l+1}, so the execution time of the entire layer is $X_{f_l} \times Y_{f_l} \times F_{l+1}$, which is the cost function subjected to memory bandwidth constraints and the number of hardware convolves available.

3.3.2. *Combined output with multiple passes.* In the case that more than FF_l input feature maps are combined to realize one output, then the aggregated output from the convolution operators may only be a partial output that must be stored in memory. The execution time is roughly the same as the time it takes to read one of the input feature maps, about $F_{l+1} \times \lceil \frac{F_l}{FF_l} \rceil$ times. This can be expressed by splitting the f_l loop and then moving the ff_l loop to the top of all loops. Hence it is possible to apply this reuse of the 2D convolvers for a fraction of the input feature maps and one output feature maps.

The number of passes required for completion is $F_{l+1} \times \lceil \frac{F_l}{FF_l} \rceil$. The execution time for completing the entire layer is $X_{f_l} \times Y_{f_l} \times F_{l+1} \times \lceil \frac{F_l}{FF_l} \rceil$. The computation starting from the m_{l-1} loop of both scenarios above can be spacially mapped in hardware.

Algorithm 4: Independent output within single pass

```
for f_l = 1 to f_l = F_l do
    for ff_{l+1} = 1 to ff_{l+1} = F_l by FF_l do
        for x_{f_l} = 1 to x_{f_l} = X_{f_l} by ΔX_{f_l}^{f_{l+1}} do
            x_{f_{l+1}} = x_{f_l} / ΔX_{f_l}^{f_{l+1}}
            for y_{f_l} = 1 to y_{f_l} = Y_{f_l} by ΔY_{f_l}^{f_{l+1}} do
                y_{f_{l+1}} = y_{f_l} / ΔY_{f_l}^{f_{l+1}}
                for f_{l+1} = ff_{l+1} to f_{l+1} = ff_{l+1} + FF_l do
                    initialize v
                    for i = 1 to i = N_i do
                        for j = 1 to j = N_j do
                            v+ = p_{f_l}(i + x_{f_l}, j + y_{f_l}) × w_{ij}^{f_l, f_{l+1}}
                    p_{f_{l+1}}(x_{f_{l+1}}, y_{f_{l+1}}) = v
```

4. **Conclusion.** In this paper, a computational model of convolution filtering and a mapping scheme for FPGA-based computing with hardware constraints are presented. The contribution of this model is to express all the storage and computational tradeoffs involved in mapping a CNN on FPGAs. Provided the model correctly captures the different tradeoffs, it can guide the proper parameterization of a CNN for a given FPGA architecture. In the future, the loop-based partition scheme will be extended to more commercial reconfigurable architectures to exploit the potential parallelism and tradeoffs in different architectures.

REFERENCES

[1] Y. L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel et al., Handwritten digit recognition with a back-propagation network, *Advances in Neural Information Processing Systems*, 1990.

[2] Y. L. Cun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proc. of the IEEE*, vol.86, no.11, pp.2278-2324, 1998.

[3] Y. L. Cun, K. Kavukcuoglu and C. Farabet, Convolutional networks and applications in vision, *Proc. of IEEE International Symposium on Circuits and Systems*, pp.253-256, 2010.

[4] S. Chakradhar, M. Sankaradas, V. Jakkula and S. Cadambi, A dynamically configurable coprocessor for convolutional neural networks, *ACM SIGARCH Computer Architecture News*, vol.38, pp.247-257, 2010.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen and O. Temam, Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, *ACM SIGPLAN Notices*, vol.49, pp.269-284, 2014.

[6] A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, pp.1097-1105, 2012.

[7] R. G. Gironés, R. C. Palero, J. C. Boluda and A. S. Cortés, FPGA implementation of a pipelined on-line backpropagation, *The Journal of VLSI Signal Processing*, vol.40, no.2, pp.189-213, 2005.

[8] F. Cardells-Tormo and P. L. Molinet, Area-efficient 2-d shift-variant convolvers for FPGA-based digital image processing, *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp.209-213, 2005.

[9] T. Chilimbi, Y. Suzue, J. Apacible and K. Kalyanaraman, Project Adam: Building an efficient and scalable deep learning training system, *The 11th USENIX Symposium on Operating Systems Design and Implementation*, pp.571-582, 2014.