Overcoming Resource Underutilization in Spatial CNN Accelerators

Yongming Shen
Stony Brook University
yoshen@cs.stonybrook.edu

Michael Ferdman
Stony Brook University
mferdman@cs.stonybrook.edu

Peter Milder
Stony Brook University
peter.milder@stonybrook.edu

***Abstract*—Convolutional neural networks (CNNs) are revolutionizing a variety of machine learning tasks, but they present significant computational challenges. Recently, FPGA-based accelerators have been proposed to improve the speed and efficiency of CNNs. Current approaches construct an accelerator optimized to maximize the overall throughput of iteratively computing the CNN layers. However, this approach leads to dynamic resource underutilization because the same accelerator is used to compute CNN layers of radically varying dimensions.**

We present a new CNN accelerator design that improves the dynamic resource utilization. Using the same FPGA resources, we build multiple accelerators, each specialized for specific CNN layers. Our design achieves 1.3x higher throughput than the state of the art when evaluating the convolutional layers of the popular AlexNet CNN on a Xilinx Virtex-7 FPGA.

I. Introduction

Convolutional neural networks (CNNs) are transforming machine learning. CNNs are being applied across diverse fields, spanning recommendation systems [1], natural language processing [2], and computer vision [3]. For CNNs, image object recognition has become the de facto benchmark, with new networks shattering all prior records in object detection and classification every year.

However, improvements in CNN accuracy are achieved through huge increases in the computational cost. State-of-the-art CNNs are already not tractable on multi-core CPUs. Although GPUs still afford adequate performance, GPU power consumption makes them significantly less attractive at datacenter scale. As a result, FPGAs offer a promising opportunity for CNN acceleration due to their programmable, massively parallel, and power-efficient computing substrate.

CNNs comprise multiple computation *layers*, whose inputs are arrays of different dimensions, given by the CNN specification. The previous state of the art for using FPGAs for CNNs is to implement an accelerator that we call a *convolutional layer processor* (CLP), which iteratively processes the layers, one at a time. A CLP design depends on parameters that control the dimensions of its computational grid; its speed depends on the compatibility of these dimensions with the CNN layers it computes. CLP parameters are jointly optimized for the ensemble of the layers in a way that maximizes the collective throughput of the accelerator.

We observe that jointly optimizing a single CLP for all CNN layers leads to dynamic underutilization of FPGA resources. Although the CLP is optimized for maximum throughput, the fixed dimensions of the CLP computational grid are sub-optimal for some, or even all, of the individual layers. On the popular AlexNet CNN [3], an "optimal" Single-CLP derived from the state-of-the-art methodology [4] has dynamic utilization of less than 75%. This means that, on average, more than one quarter of the computational resources dedicated to the CLP are unused.

To overcome this problem, we design an accelerator comprising multiple CLPs operating on a pipeline of images, with each CLP specialized for a subset of the CNN layers. Our design allows the dimensions of the CLPs to be specialized to better fit the CNN layer dimensions, improving efficiency. In this work, we demonstrate that our Multi-CLP AlexNet design can achieve 97.1% dynamic DSP utilization, improving performance by 31% over the state-of-the-art methodology when targeting a Xilinx Virtex-7 485T FPGA.

II. Resource Utilization Problem

Listing 1 presents the pseudo-code to compute a convolutional layer of a CNN. Each layer takes as input a set of N input feature maps and convolves them with filters, which are sets of previously-trained values called weights. There are M sets of filters; by convolving one set of N filters ($N \times K \times K$ weights) with the input feature maps, one of the M output feature maps is obtained. Each of the M output feature maps is computed by repeating this process with each of the M sets of filters.

A common approach for building a CNN accelerator is to construct a *convolutional layer processor* (CLP), which computes the nested loop in Listing 1. The same CLP is used to process all CNN layers, one by one. Because different convolutional layers have different dimensions (M, N, R, C, K, S), such a "one size fits all" approach creates a dynamic resource underutilization problem. In this section, we analyze how this problem affects a state-of-the-art FPGA CNN accelerator.

A. State of the Art Design

The design in [4] is the state-of-the-art FPGA accelerator for CNN convolutional layers. It employs loop transformations, such as loop reordering, tiling, and unrolling, to reorder computations and memory accesses, increasing throughput and reducing data transfer. The transformed loop is used as a template for constructing the accelerator. Critically, [4] tiles and unrolls the N and M loops (Listing 2), determining

```
I[N][(R-1)*S+K][(C-1)*S+K]  //input feature maps
O[M][R][C]                  //output feature maps
W[M][N][K][K]               //weights
for(m = 0; m < M; m++)
 for(n = 0; n < N; n++)
  for(r = 0; r < R; r++)
   for(c = 0; c < C; c++)
    for(i = 0; i < K; i++)
     for(j = 0; j < K; j++)
      O[m][r][c] += W[m][n][i][j] * I[n][S*r+i][S*c+j]
```

Listing 1. Pseudo code of a convolutional layer.

```
...
for(m=0; m<M; m+=Tm) {
  for(n=0; n<N; n+=Tn) {
  ...
    for(tm=0; tm<Tm; tm++) #UNROLL
      for(tn=0; tn<Tn; tn++) #UNROLL
      ...
```

Listing 2. Pseudo code for tiling in a convolutional layer processor [4].

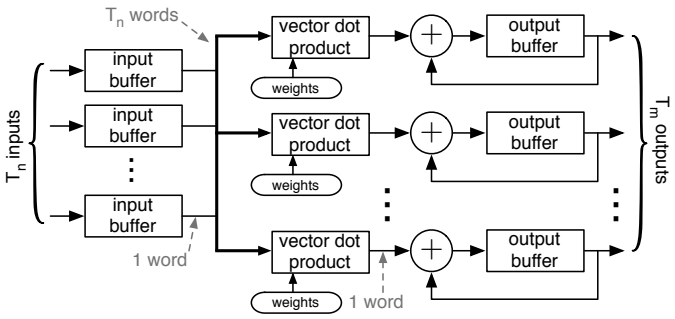

Fig. 1. A convolutional layer processor (CLP), based on the design in [4]. Each dot-product unit takes T_n inputs and T_n weights and produces one output.

the number and organization of compute units based on parameters T_n and T_m. To implement these two unrolled loops, T_m vector dot-product units are constructed, each of width T_n, each followed by an accumulation adder as shown in Figure 1. This yields $T_n \times T_m$ multipliers and adders.

Given a resource budget (e.g., a number of DSP slices), one can find the optimal T_n and T_m for a given convolutional layer. In [4], a joint optimization is performed to create a single CLP to compute *all* of the convolutional layers in the CNN. The optimization finds the (T_n, T_m) that maximize the aggregate performance of the CLP.

B. DSP Slice Utilization Problem

Although [4] produces a design optimized for the *collective performance* of all convolutional layers, we observe that its speed is limited by the fact that the different convolutional layers of the CNN have different dimensions, but all are computed on the same (T_n, T_m) CLP. Thus, the CLP that gives the best performance *across all layers* is not necessarily well suited for any one layer. Because the limiting factor of the system performance is the amount of parallel arithmetic in the CLP, a good way to quantify the cost of this mismatch is to examine the dynamic utilization of the DSP slices used in the arithmetic units. That is, we can quantify the percentage of the time that the DSP slices in the CLP are doing work versus the percentage of the time they are idle.

For example, the design in [4] targets the AlexNet CNN [3] and uses 2,240 DSP slices for floating point arithmetic. For this network and resource budget, the jointly-optimized optimum (T_n, T_m) is $(7, 64)$. However, the second layer has $(N, M) = (48, 128)$. Because 7 does not evenly divide 48, in $1/7$th of the cycles, the arithmetic units are underutilized; while computing this layer, $6/7$ths of the DSP slices remain idle. Thus, the overall dynamic DSP utilization during the evaluation of this layer on this CLP is $6/7 + (1/7) \times (1/7) = 88\%$.

Even lower utilization is observed for layers where $T_n > N$ or $T_m > M$, resulting in underutilized DSP slices on every cycle. For example, the first layer of AlexNet has $(N, M) = (3, 48)$, but because the CLP design is $(7, 64)$, the utilization is only $(3/7) \times (48/64) = 32\%$.

Analyzing all convolutional layers in [4] gives a dynamic DSP slice utilization of 74.1%, limiting the throughput to under 75% of the maximum that could potentially be achieved with the DSP slices used by this design. The cause of this utilization penalty is the mismatch between the tile parameters (T_n, T_m) and their corresponding loop sizes (N, M).

III. Multi-CLP Structure and Optimization

To improve the dynamic resource utilization and thus the performance of evaluating the CNN, we design a Multi-CLP accelerator that uses several small convolutional layer-processors rather than a single large one. The performance advantage of our design comes from the CLPs having different sizes, more closely matching the dimensions of the CNN layers. This approach is possible because CNN accelerators process many input images, allowing CLPs to concurrently work on independent inputs.

Due to the feed-forward nature of the CNN structure, it is natural to think of the layers of the CNN as a pipeline. Therefore, one way to construct an accelerator with multiple CLPs is to implement one CLP per layer. An accelerator for an L-stage CNN would have L CLPs and would operate on L independent input images. (That is, CLP1 would work on image i while CLP2 works on image $i-1$, etc.) This would have the benefit of allowing each CLP to be optimized solely for the dimensions of one CNN layer, avoiding the underutilization problem within each CLP. In this case, every CLP must still read its inputs from and write its outputs to off-chip memory, because each layer requires different data orderings than the previous layer produces, and the data sizes are typically too large to hold on chip.

A limitation of this approach, however, is that it requires the number of CLPs to be equal to the number of convolutional layers. This poses several problems for practical CNNs. First, it forces the design to divide the on-chip BRAM resources, reducing the buffer size of each CLP. As a result, the ability to exploit data reuse in each CLP diminishes, slowing each CLP

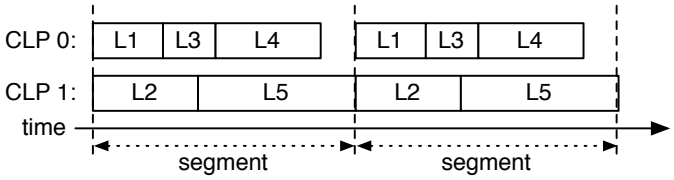

Fig. 2. Example of a Multi-CLP system with two CLPs.

and greatly increasing the whole overall memory bandwidth requirement. Second, this one-to-one mapping of CLPs to convolutional layers requires coordinating a large number of accesses to off-chip memory, which is costly in terms of performance and logic resources. Third, each CLP has an overhead cost (e.g., control logic for address calculation and loop index counting). If there are many CLPs, extra resources (especially DSP slices) must be used for control logic instead of CNN computation.

To address these limitations, we consider a Multi-CLP design with P CLPs, where $1 \leq P \leq L$ (for a CNN with L layers). Such an approach requires a single CLP to compute multiple CNN layers. The assignment of layers to CLPs is static, with each layer strictly assigned to a specific CLP. Layers assigned to the same CLP need not be adjacent in the CNN structure.

The operation timeline of a Multi-CLP accelerator is divided into *segments*. In each segment, each CLP sequentially processes its layers, with each layer having its own independent data. The segment ends when all CLPs finish processing their layers. Figure 2 shows an example where CLP0 processes three layers (L1, L3, and L4) and CLP1 processes two (L2 and L5). In each segment, each CLP only consumes data generated during the *previous* segment, avoiding data dependencies within a segment. For example, the output produced by L1 in segment i will be used as input for L2 in segment $i + 1$. This means that processing an image requires five segments of time, therefore computation from five different images will be simultaneously in flight.

There are three considerations for a Multi-CLP system to achieve high throughput:

- The convolutional layers assigned to a CLP should have dimensions compatible with the CLP dimensions to ensure high dynamic DSP utilization.
- The segment length, and thus the system throughput, is limited by the CLP that takes the longest to complete its assigned work. Ideally, each CLP is assigned work proportional to its computation speed. For example, in Figure 2, CLP0 is idle after it finishes L4 until the next segment begins.
- The on-chip memory allocated to each CLP is inversely related to the bandwidth it requires; larger CLP buffers reduce off-chip data transfer.

TABLE I
OPTIMIZED SINGLE-CLP AND MULTI-CLP ACCELERATORS.

	T_n	T_m	Layers	T_r	T_c	Cycles($\times$1000)
CLP0	7	64	1a, 1b	14	19	366
			2a, 2b	14	27	255
			3a, 3b	13	13	169
			4a, 4b	13	13	128
			5a, 5b	13	13	85
Overall						2,006

(a) Single-CLP

	T_n	T_m	Layers	T_r	T_c	Cycles($\times$1000)
CLP0	3	24	1a, 1b	28	28	732
CLP1	8	19	2a, 2b	27	27	765
CLP2	7	32	3a, 3b	13	13	338
			4a, 4b	13	13	256
			5a, 5b	13	13	170
Overall						1,531

(b) Multi-CLP

IV. EVALUATION

We demonstrate our Multi-CLP CNN accelerator design of AlexNet [3] on a Xilinx Virtex-7 485T FPGA and compare it to [4]. Leveraging the aforementioned considerations, we use an optimization method [5] to determine the best Multi-CLP design that can be achieved with the available FPGA resources.

The resulting CNN accelerator for AlexNet comprises three CLPs. CLP0 and CLP1 are responsible for processing AlexNet layers one and two, respectively. CLP2 is responsible for layers three, four, and five. We implement the Single-CLP and Multi-CLP designs using Vivado HLS 2015.4.2, and use synthesis and place and route tools to compare the two methods. We report results using single-precision floating point and a 100 MHz clock in order to compare directly with [4], although Vivado HLS can reach much higher clock frequencies for our design. Our results demonstrate that the Multi-CLP accelerator has 1.3x higher throughput compared to the Single-CLP design, with only a minor increase in the FPGA resource consumption.

Table I presents the parameters of the Single-CLP and Multi-CLP designs. T_n and T_m give the parallelism of the compute module (Figure 1) and T_r and T_c control the on-chip data tiling [4]. Because we use the design in [4] as the baseline for our CLPs, our Single-CLP system has the same parameters, $T_n = 7$ and $T_m = 64$, and the same speed, 2.0 million cycles.[1,2] Accenting the fairness of the comparison, we note that the Single-CLP and Multi-CLP designs have the same number of floating point multipliers and adders. Recall that a CLP requires $T_n \times T_m$ multipliers and adders. Our Multi-CLP

[1]The cycle counts in [4] only account for half of the convolutional layers (i.e., layers 1, 2, ..., 5, are each repeated twice per image, which we indicate as 1a, 1b, etc.). We therefore double the cycle count in Table 4 of [4] to compare with our implementation.

[2]We cannot verify T_r and T_c with prior work [4] as they are not reported.

TABLE II
PERFORMANCE AND RESOURCE USAGE FOR THE SINGLE-CLP (S-CLP) AND MULTI-CLP (M-CLP) ACCELERATORS.

	BRAM	DSP	FF	LUT	DSP Util.	Thr. img/s
S-CLP	730 (35%)	2,331 (83%)	154,813 (25%)	163,896 (54%)	74.1%	49.85
M-CLP	667 (32%)	2,469 (88%)	182,964 (30%)	182,394 (60%)	97.1%	65.32

design uses the same number ($3\times24+8\times19+7\times32=448$), but distributes them over three different CLPs.

In the Multi-CLP design, the CLPs operate concurrently on different input images. Thus the reported overall cycle count is the maximum cycle count of its individual CLPs, because this dictates the interval in which the pipelined Multi-CLP accelerator is able to start a new image. To maximize throughput, the Multi-CLP design is balanced, where each CLP takes nearly the same number of cycles to execute all assigned layers.

Table II shows the resource usage and performance of each design. We see that the Multi-CLP system provides a 1.3x throughput advantage over the Single-CLP. Because both designs use an equal number of floating-point arithmetic units, the speedup is proportional to the improvement in dynamic DSP utilization. The Single-CLP design is only able to provide useful work to the DSP slices 74.1% of the time, while the Multi-CLP design brings utilization up to 97.1%.

Table II also reports the sum of the resources used by each of the designs. These values include the CLPs only, not any platform-specific memory controllers, etc. We note that the Multi-CLP implementation uses more DSP slices compared to the Single-CLP design, even though their compute modules (i.e., the arithmetic units used for the convolution's multiplications and additions) use the same number of DSPs. This is because each CLP includes logic for address calculation and loop indexing, adding approximately 6% more DSP slices to the Multi-CLP system. Other small increases are seen in the flip-flops and LUT counts, because adding more CLPs requires additional logic beyond the DSPs and BRAMs. However, the number of DSP limits the implementations significantly more than the flip-flops or LUTs.

Further experiments [5] show that the benefits of the Multi-CLP designs increase rapidly as the FPGA size increases, growing to over 1.5x improvement on a larger Xilinx Virtex-7 and over 3x when targeting the resources that will be available in the next generation of FPGAs.

V. RELATED WORK

Although we focus on the method of [4], other recent works have proposed related types of CNN acceleration hardware, with different ways of organizing compute units or interacting with memory. For example, the 2D-convolvers used in [6]–[9] must be as large as the largest filter in any of the CNN layers; they will necessarily be underutilized when used to compute any layer that has smaller filters. As another example, in [10], the organization of the compute modules depends on the number of output feature maps and their number of rows. Because both of these parameters can change from layer to layer, an analogous resource underutilization problem occurs. In these cases, a Multi-CLP design can be employed to improve resource utilization.

VI. CONCLUSIONS

The traditional approach to FPGA-based CNN accelerator design uses a "one size fits all" approach, where a single convolutional layer processor (CLP) is used to compute all convolutional layers of the CNN. In this paper, we observed that variation in the dimensions of the CNN layers limits the throughput of the Single-CLP approach; on layers whose dimensions are a poor fit for the CLP parameters, the arithmetic units exhibit low dynamic utilization, where adders and multipliers are frequently idle.

In this work, we proposed a Multi-CLP CNN accelerator design that allows the CLP dimensions to more closely match the CNN layer dimensions, resulting in better dynamic resource utilization and higher throughput. On the Virtex-7 485T FPGA, we showed that a Multi-CLP accelerator yields a 1.3x higher throughput compared to the state-of-the-art Single-CLP design, improving the dynamic utilization of the arithmetic units from 74.1% to 97.1%.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under Grant Nos. 1533739 and 1453460.

REFERENCES

[1] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in Neural Information Processing Systems*, 2013, pp. 2643–2651.

[2] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

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

[4] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. FPGA*, 2015, pp. 161–170.

[5] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," *arXiv preprint arXiv:1607.00064*, https://arxiv.org/abs/1607.00064.

[6] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Field Programmable Logic and Applications*, 2009, pp. 32–37.

[7] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *ISCAS*, 2010, pp. 257–260.

[8] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Application-specific Systems, Architectures and Processors*, 2009, pp. 53–60.

[9] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, 2010, pp. 247–257.

[10] M. Peemen, A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Computer Design (ICCD)*, 2013, pp. 13–19.