Optimizing CNN Accelerator With Improved Roofline Model

Shaoxia Fang[1,2], Shulin Zeng[1], and Yu Wang[1]
[1]Department of Electronic Engineering, Tsinghua University, [2]Xilinx,Inc., Beijing, China
{fsx18, zengsl18}@mails.tsinghua.edu.cn, yu-wang@mail.tsinghua.edu.cn

***Abstract*— The external memory I/O bandwidth is the most common performance bottleneck for Convolutional Neural Network(CNN) inference accelerators. On the other hand, performance is also affected by many other factors such as the on-chip memory size and data scheduling strategies, making it difficult to identify the root cause of performance degradation. This paper proposes an improved roofline model specifically for the CNN accelerator, which provides a deep understanding of the bandwidth bottlenecks and points out the direction of optimization. Previous roofline models have focused on modeling and optimizing each layer, while neglecting some high-level optimizations (e.g. layer fusion and batch processing) that alleviate the bandwidth requirements. However, the uneven cross-layer bandwidth requirements can have a significant impact on the overall performance, and the combination of independently optimized layers does not necessarily result in an overall optimal solution. Our model is capable of modeling more complex data scheduling strategies and enables a larger design space than previous roofline models. We use the Xilinx CNN accelerator on ZU9 FPGA as an example for quantitative analysis and optimization. We apply the optimization method derived from the improved roofline model to the original design and ultimately achieve a 1.6x performance improvement. The derived optimization method effectively solves the severe temporary bandwidth overload problem in the original design that leads to the computational inefficiency.**

Keywords— CNN Accelerator; Roofline Model; Bandwidth Bottleneck

I. INTRODUCTION

Convolutional Neural Network(CNN)[1] inference accelerator has become a research hotspot, as the customized hardware and software co-optimization shows great advantages over general-purpose processors. A critical issue in many CNN accelerators is the external memory I/O bandwidth bottleneck, i.e. the bandwidth is not well matched to the computing power. In fact, many factors affect the performance, such as on-chip memory size, data scheduling strategies, and the characteristics of the workloads, making it difficult to identify the root cause of the performance degradation and the directions for improvement in architecture.

Once the bandwidth bottlenecks occur, increasing the I/O bandwidth or the on-chip memory size can be very costly. On the other hand, the CNN workloads typically have uneven bandwidth requirements across layers, as Fig.1 shows. Some layers are I/O bounded, while others are wasting the bandwidth. This provides architects with the opportunity to smooth out the I/O bandwidth requirements along the timeline if modeled well.

To address this problem, this paper presents an improved CNN roofline model. This work provides a deeper perspective than previous roofline model studies and allows us to model more complex data scheduling strategies.

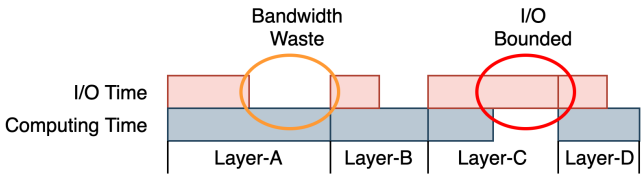

Fig. 1. The uneven bandwidth requirements across CNN layers.

First, in some previous studies, the roofline model is used to explore the optimal solution in the design space. Previous models have focused on the unrolling and tiling strategies for each layer[2][3][4][5]. Zhang et al.[2] uses a roofline model for the accelerator design exploration and attempts to find the best design for each layer in the design space. The model considers feature map reuse between tiles of one layer while ignoring the feature map reuse and the parameters reuse between layers. Furthermore, the combination of optimal designs for each layer does not necessarily lead to the optimal design for the entire model, as there are many effective inter-layer optimization techniques that can significantly improve the overall performance[6][7][8]. FINN-R[9] also provides an analysis of roofline models for typical CNN workloads with different data precisions on the same hardware platform. However, the roofline model is also simplified to two cases: 1) the parameters are stored entirely on-chip, and 2) the parameters are stored entirely off-chip.

Second, some AI benchmarks also use the roofline model for performance analysis. For example, QuTiBench[10] defines the benchmarks at multiple levels, including the architecture level and system level, and uses a roofline model for analysis. However, it has the assumption that all parameters are stored off-chip and all intermediate results are stored on-chip. In fact, in most practical cases, the parameters and feature maps are partially stored on-chip due to the on-chip memory space constraints. Therefore, data scheduling and reuse strategies are essential in order to reduce the external memory traffic that is limited by the I/O bandwidth.

Our improved roofline model is capable of modeling feature maps and parameters reuse, where the data can be stored on-chip partially as well as loaded on-chip multiple times. Batch processing and layer fusion are also considered to reflect the actual accelerator implementation. As a result, the achievable Computation to Communication Ratio(CCR) can be much higher than the previous CNN roofline model works, which leads to a larger design search space, as Fig.2 shows.

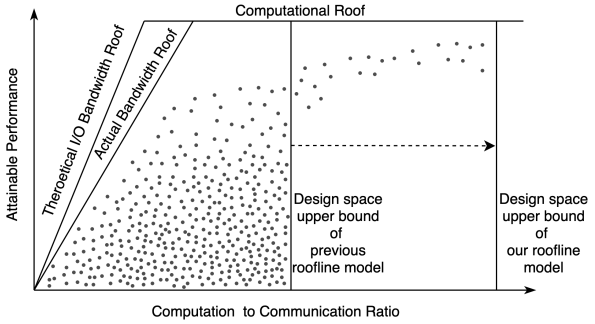

Fig. 2. The improved roofline model enables larger design space.

The main contributions of this paper are as follows:

- We propose an improved roofline model specifically for CNN, which is capable of modeling more complex data reuse strategies and enables a larger design space than previous models.
- We provide insight into the I/O bottleneck problem, analyze the imbalance in bandwidth requirements and on-chip memory requirements of a typical CNN workload, and propose a solution.
- We optimize an existing accelerator using insights derived from the improved roofline model, achieving a 1.6x performance improvement.

The rest of this paper is organized as follows: Section II introduces the background, Section III presents our improved CNN roofline model, Section IV shows the quantitative analysis and optimization of a typical CNN accelerator as well as the experiment results, and Section V concludes the paper.

II. BACKGROUND

A. CNN Accelerators

CNN inference is the feed-forward propagation of input images across layers[8], which imposes stringent requirements on the computing platform. Compared to CPUs and GPUs, FPGA and ASIC based CNN accelerators show great advantages in terms of performance, power efficiency, and cost. A famous work is the TPU designed by Google[11]. Surveys such as [8][12][13] investigate the current state of the accelerators and summarize the main techniques used.

A typical CNN accelerator structure is shown in [12]. A two-level memory hierarchy with external memory and on-chip RAM is widely used in the accelerator designs. The on-chip memory is usually divided into buffers for different purposes, such as for storing parameters, input data, intermediate data, and output data. These buffers can also be combined. Given a fixed architecture, the parameters and the feature maps may be larger or smaller than the on-chip memory size, so different data scheduling strategies are required.

B. Roofline Model

The roofline model[14] is a visual performance model used to provide performance estimates for the application running on the processor architecture. In Fig.2, there are two types of roofs: the computational roof that is limited by resource, power, and frequency; and the I/O bandwidth roof that is limited by memory hardware and access patterns.

CCR is the number of operations that can be performed per unit size of external memory access. CCR is determined by many factors: model attributes, batch number, computing engine architecture, etc. For a CNN accelerator, a higher CCR means it is more likely to achieve the computation bound. Increasing CCR reduces access to external memory and significantly saves energy[12]. Loop unrolling and tiling are effective techniques to increase CCR. [2] explores the design space by traversing all the possible loop tiling sizes and loop orders for different layers. For the cross-layer optimization, it selects an optimal unified strategy for the overall model. [3] introduces a revised roofline model for CNN that takes into account the batch parallelism. However, to the best of our knowledge, none of the previous roofline model studies have considered layer fusion which has a significant impact on CCR.

III. THE IMPROVED ROOFLINE MODEL FOR CNN ACCELERATORS

In this section, we introduce the improved roofline model for CNN accelerators. We use computing efficiency to evaluate the behavior of the accelerator on different tasks as Equation (1) shows, which indicates whether the accelerator architecture is reasonable for a given constraint.

$$ComputingEfficiency = \frac{MeasuredPerformance}{PeakPerformance} \tag{1}$$

Building a large computing engine to achieve high theoretical peak performance may be easy, whereas achieving high computing efficiency (i.e. high measured performance) is much more difficult, as the latter is affected by many factors.

Given the characteristics of the accelerator and the CNN workload, the CCR of a N-Layer CNN model can be further expressed as Equation (2). O_i is the arithmetic complexity of the i-th layer, i.e. the number of multiplication and addition operations. The external data access quantity for the i-th layer consists of 3 components: input data size D_i, which includes input model parameters and input feature maps; intermediate data size F_{mid_i}, which is the size of the temporary external data exchange during the computation of the i-th layer; output data size F_{out_i}, which is the size of the output feature map.

$$CCR = \frac{\sum_{i=0}^{N-1} O_i}{\sum_{i=0}^{N-1} (D_i + F_{mid\_i} + F_{out\_i})} \tag{2}$$

$$D_i = \alpha_i \cdot F_{in\_i} + \beta_i \cdot \frac{P_i}{B_i} \tag{3}$$

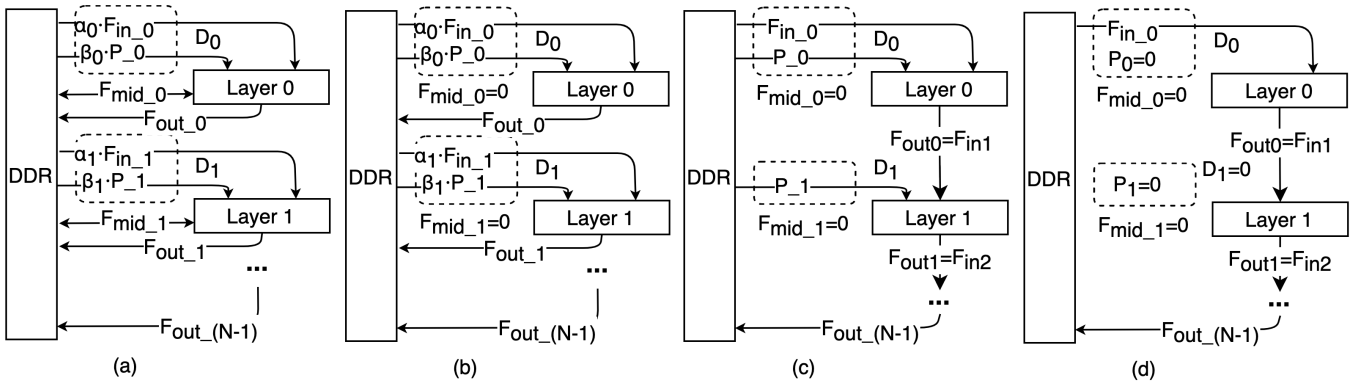

Fig. 3. The data flow of CNN inference: (a) Basic flow (b) $F_{mid\_i} = 0$ and no layer fusion (c) Full layer fusion (d) Full layer fusion and $B_i = +\infty$

For D_i, we further express it as Equation (3). F_{in_i} and P_i are the corresponding size of input feature maps and parameters. The coefficients α_i and β_i are the number of repetitions of loading the input feature maps and parameters data from external memory. B_i is the number of input feature maps that share the same set of on-chip parameters and can be thought of as the batch size processed by the accelerator. A special case is if the on-chip parameter buffer of the accelerator is large enough to permanently store all the parameters of this layer, which we can consider as $B_i = +\infty$, i.e. an infinite number of inputs share the same set of parameters. Fig.3(a) shows the basic data flow executed by the CNN accelerator.

Now consider an accelerator with the on-chip feature map buffer size of S_{FB} and on-chip parameter buffer size of S_{PB}, we define two new symbols k_{f_i} and k_{p_i} as Equation (4) and (5) to represent the minimum number of tiles for feature maps and parameters respectively.

$$k_{f\_i} = ceil(\frac{F_{in\_i}}{S_{FB}}) \tag{4}$$

$$k_{p\_i} = ceil(\frac{P_i}{S_{PB}}) \tag{5}$$

A common data scheduling strategy used in CNN accelerators is parameter stationary, where $\alpha_i = k_{p\_i}$ and $\beta_i = 1$, i.e. F_{in_i} is loaded from external memory k_{p_i} times and P_i is loaded once. We define D_i in parameter stationary strategy as D_{PSS_i} in Equation (6).

$$D_{PSS\_i} = D_i(\alpha_i = k_{p\_i},\ \beta_i = 1) \tag{6}$$

Similar is the feature map stationary strategy, where $\alpha_i = 1$, $\beta_i = k_{f\_i}$. We define D_i in feature map stationary strategy as D_{FSS_i} in Equation (7).

$$D_{FSS\_i} = D_i(\alpha_i = 1,\ \beta_i = k_{f\_i}) \tag{7}$$

D_{EM_i} is defined as the empirical maximum value of D_i in Equation (8). In the practical scenarios, mixed strategies of parameter stationary and feature map stationary can be used with various tiling methods, resulting in different α_i, β_i, and D_i values. Therefore, the D_{EM_i} is not the theoretical maximum D_i among all possible implementations of a given layer. However, D_{EM_i} is meaningful for narrowing down the range of CCR as it represents an easy-to-implement strategy that is not optimal but is effective.

$$D_{EM\_i} = max(D_{PSS\_i}, D_{FSS\_i}) \tag{8}$$

We now define CCR_{EL} as the empirical lower bound of CCR in Equation (9). For CCR_{EL}, we assume that the CNN model is computed layer-by-layer without fusion of convolution layers and $D_i = D_{EM\_i}$. Moreover, the parameters are assumed to be tiled in the direction of the output channel (which makes $F_{mid\_i} = 0$). The data flow is shown in Fig.3(b).

$$CCR_{EL} = \frac{\sum_{i=0}^{N-1} O_i}{\sum_{i=0}^{N-1} (D_{EM\_i} + F_{out\_i})} \tag{9}$$

CCR_{EU} is defined as the empirical upper bound of CCR in Equation (10). The case for CCR_{EU} is assumed to be optimal: all layers are fused. Fig.3(c) shows this case and Fig.3(d) shows the special case of $B_i = +\infty$.

$$CCR_{EU} = \frac{\sum_{i=0}^{N-1} O_i}{F_{in\_0} + F_{out\_(N-1)} + \sum_{i=0}^{N-1} (\frac{P_i}{B_i})} \tag{10}$$

Additionally, we define CCR_T in Equation (11) to represent the CCR value at the ridge point[14] of the roofline, where f is the frequency at which the MAC units operate, M is the number of MAC units, and BW_{max} is the achievable peak bandwidth of external memory.

$$CCR_T = \frac{Peak\ Performance}{Peak\ Bandwidth} = \frac{f \cdot M \cdot 2}{BW_{max}} \tag{11}$$

Obviously, CCR_T is just a property of the hardware, which is the boundary between the I/O bounded region and the computation bounded region.

IV. CNN ACCELERATOR OPTIMIZATION WITH IMPROVED ROOFLINE MODEL

In this section we choose an FPGA based CNN acceleration example system from Xilinx for analysis and optimization. The hardware specification of the system[15] is as follows:

- Device: Xilinx Zynq Ultrascale ZU9
- Accelerator IP Core: DPU-B4096
  - Frequency: 287MHz
  - MACs: 2048
  - Feature Map Buffer Size: 512KB
  - Parameter Buffer Size: 512KB
  - Quantization Format: INT8
  - Hardware Batch Size: 1
- Total Accelerator Core Number: 3
- System DDR Bandwidth: 19.2 GB/s

We choose ResNet-50 with an input shape of $224*224*3$ as the target CNN workload. This example system integrates three independent B4096 cores, each containing a private set of on-chip buffers for storing the feature maps and parameters, with no parameter sharing scheme between the cores. The peak performance of the system is 3.53 TOPS. Assuming the maximum external DDR access efficiency is 90%, then we can get $BW_{max} = 17.28GB/s$ and $CCR_T = 204$.

For the Xilinx B4096 cores, the tiling of parameters is always in the direction of output channels, so there is no intermediate state data exchange between accelerator and external memory, i.e. $FM_{mid\_i} = 0$ $(i = 0, .., N-1)$.

According to Equation(4) and (5), we can get the k_f and k_p of ResNet-50 layers on the accelerator, as Fig.4(a) shows. Fig.4(a) also shows the uneven requirement on the on-chip memory across the layers. Most layers have k_f equal to 1, which means that the feature map buffer of 512KB is sufficient for the ResNet-50 model. However, as the input image resolution increases, the 512KB feature map buffer may not be sufficient and k_f increases accordingly. In contrast, k_p is not affected by the input image resolution. It can be observed that the 512KB parameter buffer is smaller than the parameter size of most of the layers in the second half of ResNet-50.

With these specifications, the empirical lower bound CCR_{EL} and empirical upper bound CCR_{EU} can be derived.

On the other hand, the actual values of $\sum_{i=0}^{N-1} D_i$ and $\sum_{i=0}^{N-1} F_{out\_i}$ can be measured. Thus the actual CCR of the example system can be calculated, which falls in the range of empirical lower and upper bounds of CCR and is consistent with the theoretical model, as shown in Table I.

The measured throughput on the example system is 163.4 images/sec. Thus the measured performance is 1.26 TOPS and the computing efficiency is 35.7%. Although the actual CCR is quite close to CCR_T, the measured efficiency is lower than expected.

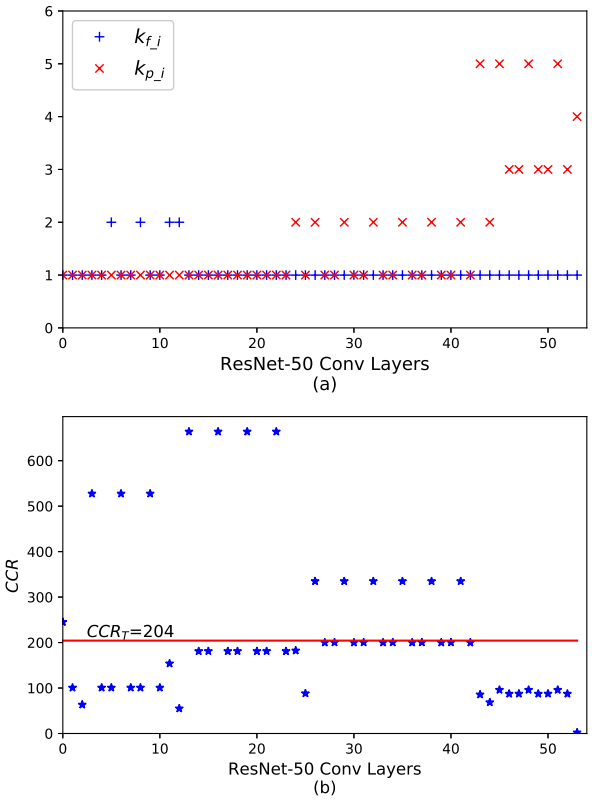

Fig. 4. (a) The uneven k_f and k_p across ResNet-50 Convolution layers. (b) The uneven CCR across ResNet-50 Convolution layers.

TABLE I
THE CCR OF DPU-B4096 EXAMPLE SYSTEM
(RESNET-50_224X224, $B = 1$)

CCR_T	CCR_{EL}	CCR_{EU}	*Actual* CCR
204	158	301	192

Considering fixed resource of hardware MAC units, on-chip buffer, and external memory bandwidth, an increase in performance means an increase in computing efficiency. In order to improve the performance, we need to optimize the architecture and improve the data scheduling strategies.

We will explore some root causes for the low computing efficiency from the perspective of bandwidth and CCR. Fig.4(b) shows the CCR distribution of the convolution layers in ResNet-50, assuming that the on-chip buffer is large enough for each layer and no tiling strategy is needed($F_{mid} = 0, \alpha = 1, \beta = 1, B = 1$). Even in this ideal case, most layers have CCR values less than CCR_T, while other have CCR values much higher than CCR_T. We can expect that when calculating the low CCR layers, there will be significant short-time DDR bandwidth requirements that may exceed the system I/O capacity and pause the hardware processing pipeline. This will result in significant performance degradation. Whereas, when computing the high CCR layers, there is almost no pressure on external memory bandwidth which is quite wasteful. This uneven distribution of CCR across layers brings about uneven use

of external memory bandwidth, which can cause temporary bandwidth overload problems and ultimately hurt the overall performance.

The improved roofline model can help us predict where the temporary bandwidth overloads will occur and how much they will be. Layer fusion is one of the effective techniques to solve the problem. However, there could be plenty of layer fusion opportunities for deep CNN models. As our roofline analysis shows, the layers with lower CCR than CCR_T should be highlighted and prioritized. Our first-priority goal is to minimize the size of the external data transfer for specific parts of the model, not the overall external data transfer size.

Suppose there are several layers named layer-0,1,2,3. Let's consider the cases of fusion of neighboring layers.

1) Firstly let's look at the case that a high CCR layer(layer-0) is followed by a low CCR layer(layer-1). Two data reuse schemes can be used. The first one is the output feature map reuse, i.e reduce both F_{out_0} and F_{in_1}, which is the common case of layer fusion. The second one is prefetching, i.e. prefetching P_1 or F_{in_1} when computing layer-0. As a result, the bandwidth requirement across layer-0 and layer-1 is smoothed.

2) The second case where a low CCR layer(layer-2) is followed by a high CCR(layer-3) layer is more complex. In most practical cases, D is the dominant component of the data transactions, so CCR of layer-3 does not benefit much from the reuse of F_{out_3}. The solution to this problem is to slice layer-3 and layer-4 and execute them in an interleaved style. Then the two schemes in the previous case can be used again.

3) For the case where both the two neighboring layers are low CCR layers, the only option for fusion is to reuse output feature map between the two layers. Batch processing is another option in such layers, if the overall latency can meet the requirement. Increasing B has a clear positive effect on such low CCR layers.

From the perspective of the CNN roofline model, we need to increase the realized CCR as much as possible (within the allowed latency), even if it is already greater than CCR_T. While a higher CCR does not necessarily lead to higher performance once the computation bounded region is reached, it gives us more opportunities to hide the memory access time and the non-convolutional computation time(e.g. pooling, element-wise, reshape, etc.) behind the convolutional computation time, and it will be easier for the data scheduling strategies to achieve higher performance.

Based on the guidance of the improved roofline model, we designed a Layer Fusion and Parameters Sharing(LFPS) optimization scheme based on the original B4096 DPU design, as Fig.5 shows. Both microarchitecture and data scheduling strategy are optimized.

We fused more than half of the low CCR layers and increased the batch size to 3($B = 3$). As shown in Fig.4(a), although the sizes of the feature map buffer and the parameter buffer are the same, the feature map and parameters have different requirements for on-chip memory.

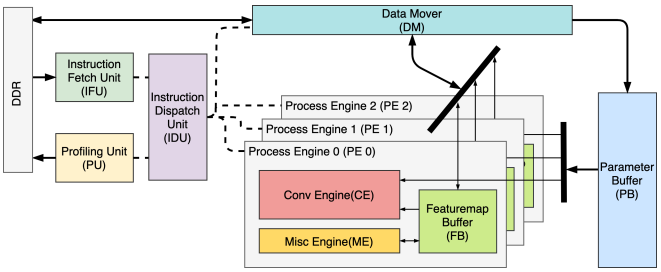

Fig. 5. The DPU optimized with LFPS .

To solve this problem, the parameter buffers of the original three independent cores are combined into a unified shared parameter buffer, which is three times the size of the original standalone buffer. As a result, the k_p per layer in ResNet-50 is significantly reduced, which also helps to increase the CCR. In this LFPS optimization scheme, the original three independent DPU cores become three synchronous processing engines sharing the same parameter buffer.

As a contrast, we also tried the Cross-Layer Optimization (CLO)[2] approach on the original design. CLO is derived from the previous roofline model, which mainly focuses on exploring all the possible loop tiling sizes and loop orders. CLO searches for the optimal tiling strategy for each convolutional layer. In order to be hardware friendly, a uniform unrolling strategy across different layers is required to fix the hardware parallelism. The global uniform solution is found by enumerating all legal solutions. The data sharing between adjacent tiles is considered, and there is no layer fusion.

We list the final performance data for the original design, the LFPS-optimized design and the CLO-optimized design in Table.II and Fig.6. Note that the theoretical peak performance, DDR bandwidth, and on-chip memory size are the same for all three accelerator designs. It can be observed that the CLO-optimized version has a larger external data transfer size and lower CCR than the original DPU design, thus resulting in lower performance than the original one. This is reasonable since the original DPU version has adopted the layer fusion strategy for some of the layers, which can be derived by comparing the size of $\sum_{i=0}^{N-1} F_{out\_i}$ with that of the unfused ResNet-50 layers. However, the global unroll parallelism found by CLO outperforms the original version, reducing the overall convolution computing time by about 10%.

The LFPS-optimized version shows significant advantages over both the original and CLO versions. Compared to the CLO version, the CCR increases by 2.6x and the performance increases by 1.6x in the LFPS-optimized version. This is an expected result, because while CLO can find an optimal global unroll strategy to minimize the sum of the executing time for each layer, it is limited by the simplified roofline model and cannot search in the larger design space that can be covered by LFPS.

TABLE II

OPTIMIZING CNN ACCELERATOR WITH ROOFLINE MODEL

Optimizations	$\sum_{i=0}^{N-1} D_i$ **(Bytes)**	$\sum_{i=0}^{N-1} F_{out\_i}$ **(Bytes)**	CCR	**Computing Efficiency**
Original	3.52E+7	5.21E+6	192	35.7%
CLO	3.59E+7	1.07E+7	166	34.6%
LFPS	1.34E+7	4.48E+6	433	57.3%

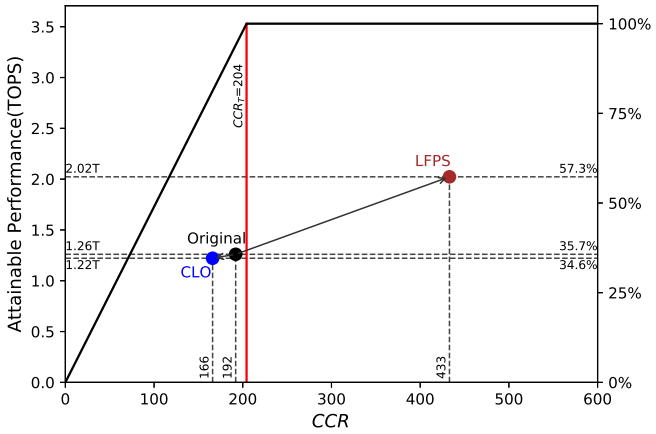

Fig. 6. The optimizations on DPU@ZU9 (workload: ResNet-50, 224x224).

V. CONCLUSIONS

In this paper, we propose an improved roofline model for CNN accelerators to address the performance degradation caused by memory bandwidth bottlenecks. Unlike previous studies that primarily focused on the modeling and optimizing each layer, our improved roofline model is capable of modeling more complex data reuse strategies and enables a larger design space.

We extend the expression of the computation to communication ratio (CCR) for CNN workloads in the roofline model. Several definitions are used to describe the effect of single-layer data scheduling strategies and the on-chip buffer sizes, while others are used to describe the layer fusion strategy and batch processing. Furthermore, we derive the empirical lower and upper bounds for CCR.

We perform a quantitative analysis using the Xilinx CNN accelerator on ZU9 FPGA as an example. Our improved roofline model provides a deeper understanding of bandwidth bottlenecks and points the way to optimization. Although the original implementation is close to the computation bounded region, we observe the uneven requirements across the layers on both external bandwidth and on-chip memory by CCR, k_f, and k_p. The severe temporary bandwidth overloads lead to computational inefficiencies.

We derive the LFPS optimization method from the improved roofline model, which provides a significant performance improvement over the original design as well as the CLO-optimized version. The CLO-optimized is limited by the design search space, resulting in a slight performance degradation compared to the original design, while the LFPS version improved the performance by 1.6x with the same hardware resources.

ACKNOWLEDGMENT

The authors acknowledge the people who contributed to the project and paper: Junbin Wang, Xi Wang, Jiangsha Ma, Taobo Wang, Cheng Chen and Yi Shan.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170, 2015.

[3] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(11):2072–2085, 2018.

[4] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. Design space exploration of fpga-based deep convolutional neural networks. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 575–580. IEEE, 2016.

[5] Paolo Meloni, Gianfranco Deriu, Francesco Conti, Igor Loi, Luigi Raffo, and Luca Benini. Curbing the roofline: a scalable and flexible architecture for cnns on fpga. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 376–383, 2016.

[6] Yu Xing, Shuang Liang, Lingzhi Sui, Xijie Jia, Jiantao Qiu, Xin Liu, Yushun Wang, Yi Shan, and Yu Wang. Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594, 2018.

[8] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, and François Berry. Accelerating cnn inference on fpgas: A survey. *arXiv preprint arXiv:1806.01683*, 2018.

[9] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth Obrien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):1–23, 2018.

[10] Michaela Blott, Lisa Halder, Miriam Leeser, and Linda Doyle. Qutibench: Benchmarking neural networks on heterogeneous hardware. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(4):1–38, 2019.

[11] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[12] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. A survey of fpga-based neural network accelerator. *arXiv preprint arXiv:1712.08934*, 2017.

[13] Ahmad Shawahna, Sadiq M Sait, and Aiman El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:7823–7859, 2018.

[14] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[15] Xilinx. Zynq DPU v3.2 IP Product Guide. https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_2/pg338-dpu.pdf. Accessed April 4, 2020.