Algorithm-hardware Co-design for Deformable Convolution

Qijing Huang*, Dequan Wang*, Yizhao Gao[1], Yaohui Cai[2], Zhen Dong, Bichen Wu, Kurt Keutzer, John Wawrzynek

University of California, Berkeley,
[1]University of Chinese Academy of Science, [2]Peking University
{qijing.huang, dequanwang, zhendong, bichen, keutzer, johnw}@berkeley.edu
gaoyizhao16@mails.ucas.ac.cn, caiyaohui@pku.edu.cn

Abstract

FPGAs provide a flexible and efficient platform to accelerate rapidly-changing algorithms for computer vision. The majority of existing work focuses on accelerating image classification, while other fundamental vision problems, including object detection and instance segmentation, have not been adequately addressed. Compared with image classification, detection problems are more sensitive to the spatial variance of objects, and therefore, require specialized convolutions to aggregate spatial information. To address this, recent work proposes dynamic deformable convolution to augment regular convolutions. Regular convolutions process a fixed grid of pixels across all the spatial locations in an image, while dynamic deformable convolutions may access arbitrary pixels in the image and the access pattern is input-dependent and varies per spatial location. These properties lead to inefficient memory accesses of inputs with existing hardware. In this work, we first investigate the overhead of the deformable convolution on embedded FPGA SoCs, and then show the accuracy-latency tradeoffs for a set of algorithm modifications including full versus depthwise, fixed-shape, and limited-range. These modifications benefit the energy efficiency for embedded devices in general as they reduce the compute complexity. We then build an efficient object detection network with modified deformable convolutions and quantize the network using state-of-the-art quantization methods. We implement a unified hardware engine on FPGA to support all the operations in the network. Preliminary experiments show that little accuracy is compromised and speedup can be achieved with our co-design optimization for the deformable convolution.

1   Introduction

Deep neural networks have achieved a series of successes in object recognition tasks, such as image classification, semantic segmentation, and object detection. Convolution, as its basic operation, is widely adopted in different neural network architecture designs. Many hardware acceleratorshave been developed to improve the speed and power performance of the compute-intensive convolutional kernels. While the use of convolution kernels for computer vision is well-established, researchers have been constantly proposing new operations and new network designs to increase the model capacity and achieve higher accuracy for various tasks. Some of these new operations lack mature and efficient support from existing hardware. FPGA, as a programmable platform, lends itself to accelerating fast-evolving deep learning algorithms. Timely and efficient hardware support for novel operations can be developed on FPGAs in weeks with high-level design tools.

Deformable convolution Dai et al. [2017], Zhu et al. [2019] is a novel and effective operation leading to the state-of-the-art accuracy for object recognition tasks. Three out of five implementations leading to top accuracy to date for object detection on the COCO dataset use deformable convolution in

their design, including the 1st-ranked model Chen et al. [2019]. Differing from the conventional convolutions with fixed geometric structure, deformable convolution samples inputs from variable offsets generated based on the input features of the target task. There are two advantages it provides to improve its modeling capacity compared to conventional convolutions: *variable sampling scales* and *variable sampling geometry*. The range for sampling at each different point varies, allowing the network to capture objects of different scales. The geometry of the sample points is not fixed, allowing the network to capture objects of different shapes. A previous study Zhou et al. [2019] has also shown that the deformable convolution network (DCN) design lies on the pareto front of the speed-accuracy tradeoff for object detection on GPUs.

There are several challenges in supporting deformable convolution on hardware accelerators. 1) there is an increased memory bandwidth requirement for loading the variable offsets. 2) the memory accesses for the input feature maps are irregular as they depend on the dynamically-generated offsets. Many existing accelerators' instruction set architecture and the control logic are insufficient in supporting the random memory access patterns. In addition, the less contiguous memory access patterns limit the length of bursting memory accesses and incur more memory requests. 3) there is less spatial reuse for the input features. Due to the variable filter offsets, the loaded input pixel for the current output pixel can no longer be reused by its neighboring output pixels. This can significantly affect the performance of the accelerators designed for output-stationary or row-stationary dataflow which leverages input reuse.

To address these challenges, we adopt an algorithm-hardware co-design approach and study the accuracy-efficiency tradeoffs for each algorithmic modification. We propose the following modifications to the deformable convolution operation to make it more hardware friendly:

1. Remove the bilinear interpolation required for calculating the final sampling address
2. Use only a square shape filter with variable dilation to reduce the total overhead from loading the offsets and to enable parallel accesses to on-chip memory
3. Limit the dynamic sampling offsets to be positive and below a fixed value to allow buffering of inputs and exploit full input reuse.
4. Convert to depth-wise convolution to reduce total FLOPs

We evaluate each modification on an FPGA System-on-Chip (SoC) that includes both an FPGA fabric and a hardened CPU core. We leverage the shared last-level cache (LLC) included in its full hardened processor system to efficiently exploit the locality from the original deformable convolution with data-dependent memory access patterns. We then implement each algorithm modification in hardware to demonstrate its effectiveness compared to the original design. With these proposed algorithm modifications, we devise a line-buffer design to efficiently support the modified depthwise deformable convolution. To demonstrate the full capability of the co-designed operation, we are in the process of training an efficient neural network model for object detection using ShuffleNetV2 as the feature extractor. We plan to quantize the network with a symmetric uniform quantizer using the block-wise quantization-aware fine-tuning process Dong et al. [2019]. Our main goals include:

1. Co-design and evaluate a new depthwise deformable convolution with hardware-friendly modifications
2. Optimize the hardware design for each algorithm modification and evaluate the performance in accelerator systems with and without an LLC.
3. Design an efficient neural network model with the proposed depthwise deformable convolution for object detection, and quantize it to ultra-low previsions.
4. Implement a hardware accelerator targeting the new network design on an FPGA SoC.

Promising results are shown from a preliminary study of algorithm-hardware co-design for deformable convolution in this paper.

2 Deformable Convolution Co-design

It is challenging to provide efficient support for the original deformable convolution on hardware due to: 1) the increased computation for the bilinear interpolation, 2) the overhead of the deformable
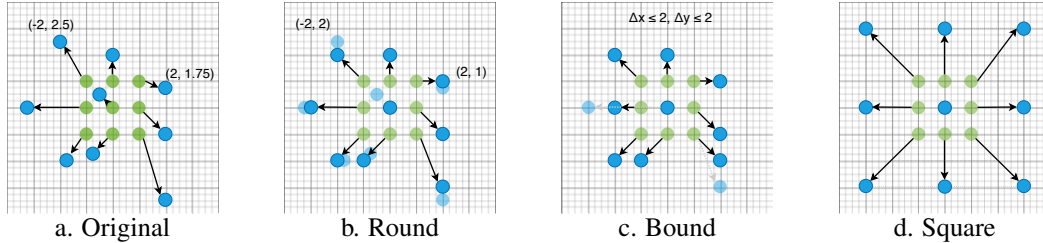
Figure 1: Major Algorithm Modifications. a) is the original deformable convolution with non-integer offsets. b) shows that the offsets are rounded to integers. c) sets an upper bound to the offsets. d) limits the geometry to a square shape.

Table 1: Notations

n	h	w	ic	oc	k	Δp
batch size	height	width	input channels size	output channel size	kernel size	offsets

offsets, 3) the dynamic and irregular input access patterns, 4) limited reuse of the input features. We perform a series of modifications to the algorithm to make it more hardware-friendly. For each modification, an experiment targeting the semantic segmentation task is run to test its impact on accuracy. We target semantic segmentation in this study as it shares the same feature extraction design but takes less time to train the end-to-end network compared to object detection. More importantly, the model capability to capture the geometric variation also affects the accuracy of semantic segmentation. We then design a specialized hardware engine optimized for each algorithmic modification on FPGA and show the performance improvement on FPGA from each modification. We demonstrate the accuracy and hardware efficiency trade-off for each modification we propose.

2.1 Algorithm Modifications

Table 2 lists all the modifications we make to the deformable convolution and the corresponding mean Intersection over Union (mIoU) for the semantic segmentation task on the Cityscapes dataset Cordts et al. [2016]. The Deep Layer Aggregation (DLA) Yu et al. [2018] network is used as the feature extractor in these experiments.

Rounded Offsets. In the original deformable design as shown in Figure 1a, the generated offsets are typically fractional and a bilinear interpolation needs to be performed to produce the final sample value from the input. Bilinear interpolation calculates a weighted average of the neighboring pixels for a fractional offset based on its distance to the neighboring pixels. It introduces at least six multiplications to the sampling process of each input, which is a significant increase ($6 \times h \times w \times ic$) to the total FLOPs. We thus round the offsets to be integers during inference to reduce the total computation. The dynamically-generated offsets are thus rounded to integers as illustrated in Figure 1b. As shown in Table 2, there is only a 0.3 drop in mIoU with this change.

Bounded Range. Another algorithmic modification to facilitate efficient hardware acceleration is to restrict the offsets to a positive range as shown in Figure 1c. This limits the size of the working set of inputs, so a fixed-size buffer can be added to the hardware to further exploit the temporal and spatial locality of the inputs. Assume a uniform distribution for the generated offsets in a 3×3 convolution kernel with stride 1, each pixel is expected to be used nine times. If all inputs within the range can be stored in the buffer, all except the first accesses to the same address will be from on-chip memory with $1 \sim 3$ cycle latency. We impose this constraint during training by adding a *ReLU6* layer after the offset generation layer to truncate offsets that are smaller than 0 or larger than N, so all offsets $\Delta p_x, \Delta p_y \in [0, N]$. Setting the bound N to 7 results in an insignificant 0.2 mIoU degradation.

Square Shape. One obstacle to efficiently supporting the deformable convolution is its irregular data access patterns, which leads to serialized memory accesses to multi-banked on-chip memory.

Deformable Network Accuracy for Semantic Segmentation on Cityscapes Dataset

Table 2: with DLA as Feature Extractor

Deformable	Round	Bound	Square	mIoU ↑
✓				79.9
✓	✓			79.6
✓	✓	✓		79.4
✓	✓	✓	✓	78.7

Table 3: with Different Feature Extractors

Feature Extractor	Operation	mIoU ↑
DLA	DeformConv	79.9
ShuffleNetV2	DeformConv	70.1
ShuffleNetV2	DeformConv + Depthwise	68.0

3

Table 4: Co-designed Hardware Performance Comparison

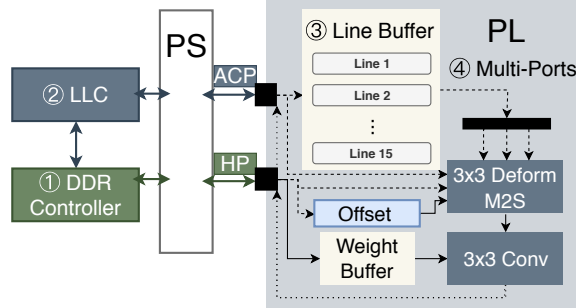| Operation	Without Deformable	Deformable	Bound (buffered)	Square (multi-ported)	Without LLC		With LLC	
					Latency (ms)	GOPs	Latency (ms)	GOPs
Full 3×3 Conv	✓				43.1	112.0	41.6	116.2
		✓			59.0	81.8	42.7	113.1
		✓	✓		43.4	111.5	41.8	115.5
		✓	✓	✓	43.4	111.5	41.8	115.6
Depthwise 3×3 Conv	✓				1.9	9.7	2.0	9.6
		✓			20.5	0.9	17.8	1.1
		✓	✓		3.0	6.2	3.4	5.5
		✓	✓	✓	2.1	9.2	2.3	8.2

Figure 2: Hardare Engine for Deformable Convolution

To address this issue, we further constrain the offsets to be on the edges of a square. Instead of using 18 numbers to represent the Δp_x and Δp_y offsets for all nine samples, only one number Δp_d, representing the distance from the center to the sides of the square, needs to be learned. This is similar to a dilated convolution with variable dilation factors for each output. This modification leads to a 0.6 decrease in mIoU. Together with all the modifications above, we achieve an mIoU of 78.7, which is only 1.2 lower compared to the original implementation.

Feature Extractor Selection. In order to further reduce the total FLOPs of the network to suit our real-time requirement, we change the feature extractor from DLA to ShuffleNetV2 Ma et al. [2018]. This results in a 9.8 decrease in mIoU, but almost ∼7× reduction in FLOPs (1.06 vs 0.146 GFLOPs). More details about ShuffleNetV2 design will be discussed in the following section.

Depthwise Convolution. We further replace all the full 3×3 deformable convolutions with 3×3 depthwise deformable convolutions and 1×1 convolutions similar to the ShuffleNetV2 building block design. This change leads to another 2.1 mIoU drop as shown in Table 3. Using the ShuffleNetV2-like structure to replace the original deformable convolutions also makes the network more uniform and easier to support on FPGAs.

2.2 Hardware Optimizations

Many hardware optimization opportunities are exposed after we perform the aforementioned modifications to deformable convolution. We implement a hardware deformable convolution engine on FPGA SoC as shown in Figure 2 and tailor the hardware engine to each algorithm modification. The experiments are run on the Ultra96 board featuring a Xilinx Zynq XCZU3EG UltraScale+ MPSoC platform. The accelerator logic accesses the 1MB 16-way set-associative LLC through the Accelerator Coherency Port (ACP). The data cache uses a pseudo-random replacement policy. Table 4 lists the speed and throughput performance for different customized hardware running a kernel of size $h = 64, w = 64, k = 256, c = 256$. In all experiments, we round the dynamically-generated offsets to integers. We use $8 \times 8 \times 9$ Multiply-Accumulate (MAC) units in the 3×3 convolution engine for all full convolution experiments and 16×9 MACs for depthwise convolution experiments.

Baseline. The baseline hardware implementation for the original 3×3 deformable convolution directly access the DRAM wihtout going through any cache or buffering. In Figure 2, the baseline implementation directly access the input and output data through HP ports and ① DDR controller. The input addresses are first calculated from the offsets loaded from DRAM. The 3×3 *Deform M2S* engine then fetches and packs the inputs into parallel data streams to feed into the the MAC units in the 3×3 *Conv* engine.

4

Caching. One hardware optimization to leverage the temporal and spatial locality of the nonuniform input accesses is to add a cache to the accelerator system. As shown in Figure 2, we load the inputs from ② LLC through the ACP port in this implementation to reduce the memory access latency of the cached values. Since the inputs are sampled from offsets without specific patterns in the original deformable convolution, the cache provides adequate support to buffer inputs that might be reused in the near future. As shown in Table 4, adding LLC results in 26.7% and 13.2% reduction in latency for the original full and depthwise deformable convolution respectively.

Buffering. With the bounded range modification to the algorithm, we are able to use the on-chip memory to buffer all possible inputs. Similar to a line-buffer design for the original 3×3 convolution that stores two lines of inputs to exploit all input locality, we store $2N$ lines of inputs so that it is sufficient to buffer all possible inputs for reuse. This implementation includes the ③ Line Buffer in Figure 2. With the effective buffering strategy, we can see in Table 4 that the latency of a bounded deformable is reduced by 26.4% and 87.5% for full and depthwise convolution respectively in a system without LLC. In a system with LLC, the reduction is 2.1% and 80.9% respectively. The depthwise deformable convolution benefits more from adding the buffer as it is a more memory-bound operation. The compute-to-comunication ratio for its input is oc times lower than the full convolution.

Parallel Ports The algorithm change to enforce a square-shape sampling pattern not only reduces the bandwidth requirements for loading the input indices in hardware, but also helps to improve the on-chip memory bandwidth. With non-predictable memory access pattern to the on-chip memory, only one input can be loaded from the buffer at each cycle if all sampled inputs are store in the same line buffer. By constraining the shape of deformable convolution to a square with variable dilation, we are guaranteed to have three different line buffers with each storing three sampled points. We can thus have three parallel ports (④ Multi-ports in Figure 2) accessing different line buffers concurrently. This co-optimization improves the on-chip memory bandwidth and leads to another $\sim 30\%$ reduction in latency for depthwise deformable convolution.

3   Conclusion

In this work, we perform a detailed accuracy-efficiency trade-off study for each hardware-friendly algorithmic modification to the deformable convolution operation, with the goal of co-designing an efficient object recognition network and a real-time embedded accelerator optimizing for accuracy, speed, and energy efficiency. Experimental results show that these modifications lead to significant hardware performance improvement with little accuracy loss.

References

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9308–9316, 2019.

Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Hybrid task cascade for instance segmentation, 2019.

Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points, 2019.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. *arXiv preprint arXiv:1905.03696*, 2019.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2403–2412, 2018.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv:1807.11164*, 2018.