A Real-Time 1280 × 720 Object Detection Chip With 585 MB/s Memory Traffic

Kuo-Wei Chang, Hsu-Tung Shih, Tian-Sheuan Chang, Senior Member, IEEE, Shang-Hong Tsai, Chih-Chyau Yang, Associate Member, IEEE, Chien-Ming Wu, Member, IEEE, and Chun-Ming Huang, Member, IEEE

Abstract—Memory bandwidth has become the real-time bottleneck of current deep learning accelerators (DLAs), particularly for high definition (HD) object detection. Under resource constraints, this article proposes a low memory traffic DLA chip with joint hardware and software optimization. To maximize hardware utilization under memory bandwidth, we morph and fuse the object detection model into a group fusion-ready model to reduce intermediate data access. This reduces the YOLOV2's feature memory traffic from 2.9 to 0.15 GB/s. To support group fusion, our previous DLA-based hardware employees a unified buffer with write-masking for simple layer-by-layer processing in a fusion group. When compared to our previous DLA with the same processing element (PE) numbers, the chip implemented in a 40-nm process supports 1280 × 720 at 30 frames per second (FPS) object detection and consumes 7.9× less external dynamic random access memory (DRAM) access energy, from 2607 to 327.6 mJ.

Index Terms—Deep learning accelerator (DLA), high definition (HD), layer fusion, object detection.

I. INTRODUCTION

OBJECT detection with deep learning has attracted significant research attention in recent years due to its wide success over traditional computer vision methods [1], [2]. However, with deeper and wider deep learning models and larger input size, real-time model execution poses challenges of high computation cost and memory bandwidth, especially for edge devices as in autonomous driving. Thus, hardware acceleration with deep learning accelerators (DLAs) is required to tackle these challenges.

Many DLAs have been proposed with massive parallel processing elements (PEs) to solve the high computation cost and different data reuse policies to reduce memory traffic. We can classify these designs into layer-by-layer processing or layer fusion processing according to their hardware scheduling. The widely used layer-by-layer DLAs process one layer after another, which only needs to store one layer of weights and partial feature maps inside a chip. They reduce memory traffic by different data reuse [3]–[5], larger on-chip dynamic random access memory (DRAM) macro [6]–[10], precision adaptive design [11]–[13], and sparse convolution [14]–[17]. In which, major external DRAM bandwidth savings come from reusing input, weights, or partial sums of output [3]. Similar approaches have also been adopted in DLAs specific to object detection [18]–[20]. In addition, most of these object detection designs use abundant memory resources in field-programmable gate array (FPGA) [18] to avoid frequent external DRAM access, which does not apply to application specified integrated circuit (ASIC) designs for edge devices. These data reuse policies are limited to one-layer processing only. All these layer-by-layer DLAs have to save per layer output to the external DRAM and load it back for next layer processing, which causes high memory traffic and inhibits further possible processing speedup due to lack of data.

In contrast, layer-fusion DLAs [21]–[23] fuse multiple-layer computation that processes the next layer once its input data are ready. Thus, it only needs to load the input data of the first layer and save the output data of the last layer in the fusion group and can save the access of intermediate data. The fusion requires a large buffer to store intermediate feature maps for fusion processing. To avoid this, the input is usually partitioned into tiles for processing. However, the overlapped areas between tiles have to be stored and recomputed due to data dependency, which results in significant buffer cost when more layers are fused. Thus, nonoverlapped tile processing proposed by split-convolution neural network (CNN) [24] or block convolution [25] avoids this without considering the overlapped area and still achieves similar accuracy as the original model. Previous nonoverlapped tile processing is targeted on GPU or FPGA, which assumes a large enough buffer for weight storage. However, DLA with ASIC used in edge devices has a much smaller buffer. Note that the layer fusion assumes that all weights within the fusion group should be stored on the chip to avoid repeatedly access during tile processing. This poses design difficulties for a modern network, whose per layer weight number could be easily over 1 M, exceeds buffer size, and thus inhibits the possibility of layer fusion. Such a problem cannot be solved by hardware design only as in previous approaches, which needs model adaption for better algorithm and architecture codesign.

This work offers a fusion-ready hardware and software with collaborative optimization to develop a low memory
traffic DLA for high definition (HD) size object detection to address the aforementioned difficulties. The baseline model, YOLO-v2 [1], needs 55.6 M parameters with external memory traffic of 1.6 GB weight and 4.6 GB feature map for $1280 \times 720$ at 30 frames per second (FPS) execution. We propose the resource-constrained network fusion and pruning (RCNet) to make the model group fusion ready. The preceding process has also combined hardware-specific guidelines to make the model hardware-friendly. This allows the model to fit within the limits of weight and feature buffer size, maximizing the benefits of layer fusion and hardware utilization. Following model optimization, the hardware based on our prior design [5] uses the unified buffer design with write masking to support group fusion. Its computing flow also employs the modified nonoverlapped tile processing [24], [25] to reduce the data dependency between tiles. The final chip is implemented with the Taiwan Semiconductor Manufacturing Company (TSMC) 40 nm process, which achieves real-time HD object detection and needs $7.9 \times$ lower external DRAM energy than our previous design [5] with the same PE numbers.

The rest of the article is organized as follows. Section II introduces the proposed RCNet for YOLO-v2. Section III presents the proposed architecture. The implementation results and comparisons are shown in Section IV. Finally, this article is concluded in Section V.

II. RCNET FOR YOLO-V2

A. Overview

The main limits for maximizing the benefits of layer fusion are the weight and feature map buffer sizes. If the weight buffer cannot hold all weights in a fusion group, weights must be read repeatedly from external memory for each tile processing, resulting high memory traffic. The tile size is determined by the size of the feature map buffer, which is a tradeoff between accuracy and hardware cost.

For object detection, we choose YOLO-v2 as our baseline model for its simplicity. YOLO-v2 needs 55.6 M parameters, which makes layer fusion very unfeasible since each layer parameter could easily exceed the weight buffer capacity of edge devices. To make it fusion-ready while maximize hardware utilization, we propose using RCNet to morph the model according to the buffer size. This resource-constrained YOLO-v2 is denoted as RC-YOLOv2.

B. Lightweight Model Conversion

The backbone of the original YOLO-v2 is a simple stack CNN structure like VGG [26]. This model is not ready for layer fusion due to its large model size. For a fusion-ready model, the weight numbers of any two consecutive layers shall be smaller than the weight buffer size after RCNet. Otherwise, the fusion operation will be degenerated to a layer-by-layer one.

To make YOLO-v2 fusion ready, we replace the basic convolutional layer with the MobileNetv2 [27] block that combines one depthwise, two pointwise convolutions, and skip connection layers as shown in Fig. 1(a). In this work, inspired by Radosavovic et al. [28] that the expansion factor used in MobileNetv2 is not a must, we remove the first pointwise layer of MobileNetv2 as shown in Fig. 1(b). Other model compression approaches can also be applied to reduce model size. This step can be skipped if the input model is near fusion-ready. Our RC-YOLOv2 is constructed by stacking the blocks shown in Fig. 1(b).

C. Buffer Size Constrained Structure Morphing

Even after lightweight conversion, directly fusing multiple layers as in [25] cannot maximize the number of fused layers due to the uneven distribution of per layer size. To solve this, we propose a new pruning procedure inspired by Gordon et al. [29] to maximize the number of fused layers while remaining within the weight buffer size set by designers. Fig. 2 shows the detailed procedure. We first decide the group partition for fusion by analyzing the model from input to output. If the size of a layer exceeds the size of the available weight buffer, the fused layer group ends from its previous layer and we start a new fusion group from this layer. Our goal is to create a network in which the total weight size of each fusion group is close to the weight buffer size $B$, and to fuse more layers to avoid external memory traffic for intermediate data.

1) Formulation: The goal of training any network is to minimize a loss $L$

$$
\min_{\theta} L(\theta)
$$

where $\theta$ denotes all trainable parameters of the network and $L$ is a loss measuring how well a network is and depends on the target task. Let $O$ denotes the network after model conversion. In the first step, we partition the network into several fusion groups $G = \{G_1, G_2, \ldots, G_k\}$. As mentioned above, the fusion strategy is from input to output. In this step, we allow the total weight size of each fusion group to exceed the weight buffer size in a certain range (50% for example in this work). In the second step, the total weight size of the fusion group will be slimmed to be smaller than the weight buffer size. At the same time, the new network $O'$ learns the optimal structure with these fixed fusion groups. The formulation is as below

$$
O' = \arg \min_{F(G_i) \leq B} \min_{i=1-k} L(\theta)
$$
where $F$ represents the total weight size of each fusion group. To reduce the weight size of each fusion group, we use L1 regularization to train the scaling factor $\gamma$ in batch normalization (BN) that is applied after every convolutional layer and set the model size as a constraint. Therefore, the optimization problem is as below

$$\min L(\theta) + \lambda \delta(\theta)$$

(2)

where $\lambda$ denotes the regularization parameter, and $\delta(\theta)$ denotes the regularization term constrained by the weight size. Assume the weight size of a convolutional layer is $S_i$. We can rewrite the weight size as

$$F(\text{layer } L) = S_i \sum_{i=0}^{C_i^{\text{in}}-1} \sum_{j=0}^{C_i^{\text{out}}-1} B_{L,i,j}$$

(3)

where $A_{L,i}(B_{L,i,j})$ is the indicator function that is zero if the $i$th input ($j$th output) of the layer $L$ is zeroed out. $C_i^{\text{in}}$ and $C_i^{\text{out}}$ are the input and output channel for the $L$th layer, respectively. Therefore, the regularization term of the layer $L$ becomes

$$\delta(\theta, \text{layer } L) = S \sum_{i=0}^{C_i^{\text{in}}-1} |\gamma_{L-1,i}| \sum_{j=0}^{C_i^{\text{out}}-1} B_{L,i,j}$$

$$+ S \sum_{i=0}^{C_i^{\text{in}}-1} A_{L,i} \sum_{j=0}^{C_i^{\text{out}}-1} |\gamma_{L,j}|.$$  

(4)

Then, the whole regularization term becomes

$$\delta(\theta) = \sum_{L=1}^{N} (\theta, \text{layer } L)$$

(5)

where $N$ represents the number of layers. After training, we prune the channel with the smallest $\gamma$ value in each fusion group to satisfy the weight buffer size constraint.

To avoid a long training time, we adopt the pruning-from-scratch method from [30]. Instead of $\gamma$ values, all other trainable parameters like weights in the convolutional layer are frozen for training and set as random values. We only train the $\gamma$ value under fixed random weights. Therefore, (1) becomes

$$O' = \min_{F(G(\gamma)) \leq B} \min_{i=1-k} L(\gamma_i)$$

(6)

and (2) becomes

$$\min_{\gamma} L(\gamma) + \lambda \delta(\gamma).$$

(7)

These can speed up the training procedure significantly without performance degradation. We only have to train the entire network with all trainable parameters once when we finally obtain the low memory traffic model.

This procedure is done iteratively. Once going through one iteration, the total model size of the network will become smaller. The procedure can be stopped at any time when meeting the requirements or when encountering a dramatic accuracy drop. In which, to avoid the new network bounded by its original shape, at the first few iterations, we use uniform channel width scaling to scale the entire network to its original model size after the second step.
affect hardware utilization significantly due to much smaller input. However, the first input layer will not be affected by this significantly due to its larger input. Besides, the group partition within a residual block will cause extra data access due to the shortcut connection. Thus, based on the above observations, we combine the following hardware-oriented fusion guidelines with the RCNet to make the hardware fully utilized and reduce unnecessary traffic for YOLOv2.

1) **The First Layer With Downsampling Shall be Fused With Other Layers to Increase Utilization:** For computer vision tasks, the image input comprises only three input channels. The first convolutional layer usually follows by a pooling layer or uses strides to reduce the computation cost. To improve the benefits of layer fusion in the first fusion group, we will ignore the downsampling of the first layer and fuse the first layer with others according to buffer size constraints. Moreover, due to only three channels in the first layer, the tile size can be maximized and stored in a unified buffer. Thus, PE utilization can still be kept high even after the pooling layer.

2) **A Group of Fused Layers Shall Have No More Than Two Downsampling Layers:** The downsampling layers such as the pooling layer or stride will reduce the feature map size. However, too many pooling layers will cause too small feature maps to fully utilize parallel PEs. Thus, as a design tradeoff for our current design, we limit no more than two downsampling layers. This also applies to the convolution with strides.

3) **A Residual Block Shall be in the Same Group of Fused Layers:** The input data of the first layer in a residual block needs to be stored due to the skip connection. To avoid DLAks accessing the skip input from DRAM repeatedly, we prefer that all layers in a residual block must be in the same group of fused layers.

Of the three guidelines, the first two are for hardware utilization. If they are not met, the hardware utilization will become lower and thus result in lower throughput. The final one is for memory bandwidth. If we cannot fuse the layers within the residual connections, extra external DRAM access will be needed. The impact depends on the size of the feature input in that residual block. In general, the data access of the feature input will be one time than that with fusion.

**III. System Architecture**

**A. Overview**

Fig. 5 shows the architecture of the proposed DLA based on our prior design [5]. This design is a typical systolic array type with tile-based scheduling. It has a weight buffer for storing fusion weights and a unified buffer with left and right buffers for storing the feature maps. One PE block has an $n \times 3$ MAC array design. The PE block has $n$ inputs broadcasted horizontally in the same input column vector, and three weights broadcasted vertically in the same weight column vector to optimize for $3 \times 3$ convolutions. The multiplication results are summed along the diagonal direction. Finally, the partial sum of PE blocks will be accumulated at the accumulator to generate the output. To support the HD size object detection, this DLA core has 768 multiplier accumulator (MAC) units, which are split into eight PE blocks. Each PE block consists of a $32 \times 3$ MAC array with 32 feature inputs and three weight inputs.

**B. Hardware Support for Layer Fusion Group**

To support layer fusion, the original design has two major architecture changes: a larger weight buffer for fusion group weights and a unified buffer for fusion group execution. Besides, the nonoverlapped tile processing from [25] is also used in the computing flow. The weight buffer size is set to 96 KB after several experiments as shown in Section IV. Nonoverlapped tiling partitions an input into nonoverlapped tiles and processes them with constant boundary extension or zero padding. This can help to avoid data dependency between tiles and make layer fusion easier.

However, for a small tile size, the accuracy loss is greater. In this work, the tile size is determined by the on-chip feature buffer size, as opposed to the *ad hoc* fixed value in [25]. For a fusion group, we first assume the required input feature map size (map) and then calculate the output feature map size for all layers in a fusion group based on the channel number and buffer size, i.e., map/Pooling Factor $\times$ channels $=$ Buffer Size. Then we select the smallest map size value as the input map size of this group to ensure that such size is smaller than the buffer size for each layer. Assume Map Size $=$ Tile Width $\times$ Tile Height. Then, the width of the tile is set to be the same as the width of the feature map to avoid padding on the left and right boundaries of a tile. The top and bottom boundaries of a tile use boundary extensions for the nonoverlapped tile processing. The tile height can then be set as the maximum allowable value. This tile processing is applied to the complete model.

The layer fusion execution in nonoverlapped tile processing is simply layer-by-layer processing within a fusion group, but all intermediate data is from the internal buffer rather than DRAM. Thus, the input and output buffers in the original design now act as a ping-pong buffer, which is denoted as the unified buffer as the left and right buffers shown in Fig. 5. When the left buffer acts as the input buffer, the right buffer will act as the output buffer. Then, after the convolution has been completed, their roles as input and output are switched. Thus, all intermediate data are from the local buffers.
Fig. 5. (a) Proposed system architecture and (b) its dataflow.

Fig. 6. (a) Illustrations of input and output maps and output generating order of output data reuse. (b) Data order for the input and output buffers without any transposed addressing. (c) Data order for the input and output buffers with transposed addressing by the write-masking in SRAM.

However, the input buffer is addressed along the spatial dimension, whereas the output buffer is addressed along the channel dimension. Because of this addressing inconsistency, it is inconvenient to use a direct I/O merging buffer, which has not been addressed in [25]. To address this issue, one convenient solution is to use the write-masking capability of the on-chip static random access memory (SRAM) to reorder the output data to match the desired input data.

This transposed addressing is illustrated in Fig. 6. To support such access without extra overhead, we divide the data words into eight banks [e.g., A1–A8 in one bank, B1–B8 in another bank in Fig. 6(b)] and use the byte write capability of the SRAM to write OA1–OH1 to different banks as in Fig. 6(c). This can reorder the output to be the expected input order. This helps smooth layer fusion to avoid complex control and provide enough data amount.
IV. EXPERIMENTAL RESULT

A. Resource Constrained YOLOv2 for HD Size Object Detection

The RCNet is applied to our target HD size object detection, YOLO-v2 [1]. This article adopts the original YOLO-v2 model as our baseline, which is trained with the Pascal VOC 2007 + 2012 dataset with 74.23% mAP evaluated on the Pascal VOC 2007 dataset [31]. The training uses stochastic gradient descent (SGD) with weight decays. For the learning rate, we use the warm-up strategy from 0 to 0.1 in the early epochs and then the step decay policy until 0.0001. We use L1 loss to regularize the scaling factor \( \gamma \) in BN. The final model is pretrained on ImageNet and then trained on this dataset.

The baseline model has 55.6 M model parameters and needs 98 MB feature map I/O per inference, which will be 2.9 GB/s for 30 FPS. With the proposed approach, the new model, RC-YOLOv2, is shown in Fig. 7. We can reduce the model parameter to 1.014 M with 72.12% mAP and 5.01 MB feature map I/O under the constraint of 96 KB weight buffer size. The model size is reduced by 98.7% (55.6–1.014 M) and 73.3% (3.806–1.014 M) when compared to the baseline model before and after model conversion, respectively.

The channel number of the shortcut and 1 × 1 convolution path will be inconsistent for the summation of the residual block after the RCNet pruning. To solve this issue, our higher priority is to keep the channel number from the 1 × 1 convolution and sum with the block input of the same number of channels. We discard extra channels from the block input as shown in Fig. 8(a). If the channel number of the block input is smaller than the convolutional output, extra channels from the convolutional output are output directly as in Fig. 8(b).

The RC-YOLOv2 is also trained and tested on an HD size dataset, IVS_3cls [32] for object detection on road traffic, as shown in Table I. The input size is 1920 × 960. The original model has 88.2% mAP, 55.6 M model size, and 131.62 MB feature map I/O. With the proposed model conversion, the model size is significantly reduced, but the external I/O still needs 130.65 MBs. A naive fusion for 100 KB weight buffer size only fuses a small fraction of layers and still needs 80.45 MB I/O. With RCNet, the result model can achieve 80.81% mAP, 1.76 M model parameters, and 21.15 MBs feature map I/O. Further quantization to 8-bit does not affect accuracy. The accuracy drop can be recovered by pretraining on ImageNet before training on this dataset. The proposed RCNet can be applied to other tasks as well, like semantic segmentation as shown in Table II and image classification as shown in Table III.

In RCNet, an important constraint is the size of the weight buffer. Different buffer sizes will lead to different network structures. Overall speaking, a larger buffer will need smaller feature I/O and has higher accuracy since the constraint caused by the buffer size is smaller. However, a larger buffer size means a higher cost for deep neural network (DNN) accelerator design. To study this effect, Fig. 9 shows the weight buffer size effect on RC-YOLOv2 for the target total model size around 1 M. Feature I/O goes higher with a smaller buffer size. When the buffer size is under 100 KB, the mAP drop will be significant. Therefore, in this work, we select the weight buffer size around 100 KB, which is 96 KB based on the selected PE numbers. Fig. 10 shows the results for different final model sizes under 100 KB weight buffer. The size of the network can be reduced to about 1 M within 3% mAP drop. We can
TABLE III
ABLATION STUDY OF VGG16 [35] ON THE IMAGE NET
DATASET [36] FOR 200 KB WEIGHT BUFFER SIZE

| Conversion Only? | Naive Fusion? | RCNet?   | Quantization? |
|-----------------|--------------|---------|--------------|
| VGG16           | ✓            | ✓       | ✓            |

Fig. 9. RC-YOLOv2 under different weight buffer sizes.

also obtain similar results for different weight buffer sizes. Therefore, we select 1 M as our final model size target.

B. Hardware Implementation Result

Fig. 11 shows the chip photograph and performance summary, which is fabricated with a TSMC 40-nm CMOS process. It occupies 4.56 mm² with 480 KB SRAM, including 384 KB unified buffer and 96 KB weight buffer. The peak performance is 460.8 GOPS at 300 MHz for full PE utilization. This chip can execute this object detection model at 30 FPS for 1280 × 720 HD images and 20 FPS for 1920 × 1080 full HD images. The core power consumption is 692.3 mW, measured by running the RC-YOLOv2.

Fig. 12 shows the channel number and data bandwidth of each layer for the RC-YOLOv2 model targeted to HD (1280 × 720) image input. This figure also shows the groups of fused layers separated by red dashed lines, which are usually at the pooling layer. Group 1 comprises a 3 × 3 convolution with pooling and two residual blocks with pooling. This group follows the above guideline on the first layer since the image input only has three input channels that will cause low hardware utilization. Besides, the weight size of the first group is small, and the map size is large. Thus, more fused layers in this group can reduce a large amount of bandwidth in this model. In addition, the second and the fourth fusion groups are separated by only one pooling layer since more pooling layers in these groups will cause low hardware utilization. Other groups are decided by fusing as many layers as possible to fit the weight buffer size. The data bandwidth is shown as the yellow line chart in the figure, which includes all feature map I/O and weight access. With the group fused layers, the layer-by-layer external memory traffic can be reduced by 37%–99% for different layers.

Table IV shows data bandwidth comparison of the RC-YOLOv2. The proposed approach can save 85%, 87% of memory traffic for image size 416 × 416, and HD (1280 × 720) with 30 FPS, respectively, which is 6.5× and 7.9× reduction. Larger inputs will be benefited more from the fused layer.

This required bandwidth easily falls within the range of DDR3 DRAM bandwidth (12.8 GB/s).

Fig. 13 shows latency and memory bandwidth under different weight buffer size constraints to execute RC-YOLOv2 model on full HD images. With a larger buffer size, more layers can be fused for lower memory bandwidth, reducing 38% bandwidth from 50 to 200 KB buffer. The reduction is saturated for 300 KB buffer size because it already has the maximum group of fused layers for our lightweight model.

C. Area and Power Analysis

For the presented chip, on-chip SRAM occupies 63.9% of the area due to the large size of the unified buffer and weight buffer for layer fusion. For the logic area, the PE array occupies 42% of the area due to 768 MAC. The accumulator occupies 28% of the area due to 24-bits adders and few FIFO registers for partial sum accumulation. The controller occupies 21% of the area due to a large number of multiplexers and wire routing between the buffer and the DLA core for layer fusion processing.

Fig. 14 shows the power breakdown of the chip. Memory access accounts for 51% of power due to the large amount of access for layer fusion. However, we only need to access the original feature map and its output feature maps once through I/O pads. Thus, I/O pads consume 13.4% of the total power. The combinational logic consumes 19.5% of power due to 768 PEs, the pipelined accumulator, and the processing of BN and ReLU6. The register logic consumes 13.7% because most of the data are stored in on-chip memory, and thus needs
few registers for pipeline hardware and accumulation. This also results in lower power consumption in the clock network, which accounts 2.2% of total power. The core energy for 1280 × 720 at 30 FPS is 629.3 mJ.

When comparing with our prior design [5] with the same PE numbers and model, the area overhead due to group fusion is the larger buffer. However, if the buffers for both designs are partitioned into the same-sized banks, the energy consumption of these two designs will be comparable due to the same internal memory access amount and computation.

Above power consumption is only for the chip itself. The main contribution of this design is the low external memory traffic. The energy of the external DRAM access is reduced by 7.9 ×, from 2607 to 327.6 mJ, as shown in Table IV, when compared to our prior design [5] with the same PE numbers and model.

### D. Design Comparison

Table V shows comparisons with other designs, which is difficult due to different tasks and fusion design.

The peak throughput of the proposed design can reach up to 460.8 GOPS. The area efficiency is 0.25 GOPS/KGE and 101.05 GOPS/mm². The area efficiency is better than most of the designs except [22] due to its 7 nm process.
The core power consumption is 692.3 mW. The core power efficiency can reach up to 0.66 TOPS/W because of the simple and regular data flow. Our power efficiency is higher than [3], [14], [22] but lower than Envision [11], SRNPU [23], and THINKER [12] after normalized power efficiency. Envision [11] has higher power efficiency only when using lower precision hardware to compute 30%–60% sparse network at lower supply voltage. These low power techniques can be applied to our design as well if needed. Thinker [12] saves SRAM access power by using area-hungry flip-flops and thus has much higher area cost than our design. However, both Envision and THINKER are nonfusion designs. Thus, their external DRAM access will be similar to our prior design with the same PE numbers when executing the same model, which is significantly higher than our design and will cancel out the benefits of the lower chip power consumption.

For fusion-based designs, SRNPU [23] designs a small dedicated network for super resolution. The total model size is around 130 K, which can be stored on chip. This is not easily transferable to other networks. It has higher power efficiency than our design, but its area efficiency is much lower due to the large amount of small size cache accessing, more complex routing, and larger buffer area. Lin et al. [22] used mega scale on-chip buffer for fusion and has lower power efficiency than our design. Both designs use the direct layer fusion, which cannot fully exploit the benefit of layer fusion. Besides, they do not consider hardware utilization for fusion.

In summary, the proposed fusion design has achieved competitive performance for our core design compared to others. In addition, our design saves significant external DRAM energy and has higher area efficiency than others.

V. CONCLUSION

This article proposes a deep learning chip with a low memory bandwidth for real-time HD object detection. We use group fusion to solve the high external memory traffic. The proposed RCNet, as well as the unified buffer design, enable this fusion. When compared to the previous naive fusion approach, this approach reduces the external memory traffic from 80.45 to 21.55 MB for 1920 × 960 input with a weight buffer size of 100 KB. This fusion also included hardware-oriented guidelines to maximize hardware utilization and reduce the bandwidth by 7.9 × to 585 MB/s for HD images with 72.12% and 80.81% mAP for the Pascal VOC 2007 and HD size datasets, respectively. The final chip implemented on a TSMC 40-nm CMOS process can execute real-time object detection at 1280 × 720 at 30 FPS while reducing the external DRAM access energy from 2607 to 327.6 mJ. The proposed method can also be easily integrated into other existing DLAs to improve energy consumption.

REFERENCES

[1] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, Dec. 2017, pp. 6517–6525.
[2] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, “A survey of deep learning-based object detection,” IEEE Access, vol. 7, pp. 128837–128868, 2019.
[3] Y.-H. Chen, T. Krishna, J.-S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” IEEE J. Solid-State Circuits, vol. 52, no. 1, pp. 127–138, Nov. 2016.
[4] F. Tu, S. Y. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, “Deep convolutional neural network architecture with reconfigurable computation patterns,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 8, pp. 2220–2233, Aug. 2017.
[5] K.-W. Chang and T.-S. Chang, “VWA: Hardware efficient vectorwise accelerator for convolutional neural network,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 67, no. 1, pp. 145–154, Jan. 2020.
[6] T. Chen et al., “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” SIGARCH Comput. Archit. News, vol. 42, no. 1, pp. 269–284, Feb. 2014.
[7] Y. Chen et al., “DaDianNao: A machine-learning supercomputer,” in Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Cambridge, U.K., 2014, pp. 609–622.
[8] Z. Du et al., “SiHianNao: Shifting vision processing closer to the sensor,” in IEEE Int. Symp. Comput. Archit. (ISCA), Portland, OR, USA, 2015, pp. 92–104.
[9] D. Liu et al., “PuDianNao: A polyvalent machine learning accelerator,” ACM SIGARCH Comput. Archit. News, vol. 43, no. 1, pp. 369–381, Mar. 2015.
[10] T. Chen et al., “A high-throughput neural network accelerator,” IEEE Micro, vol. 35, no. 3, pp. 24–32, May 2015.
[11] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, “14.5 en-
vision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI,” in IEEE ISSCC Dig. Tech. Papers, San Francisco, CA, USA, Feb. 2017, pp. 246–247.
[12] S. Yin et al., “A high energy efficient reconfigurable hybrid neural network processor for deep learning applications,” IEEE J. Solid-State Circuits, vol. 53, no. 4, pp. 968–982, Apr. 2018.
[13] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, “UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision,” IEEE J. Solid-State Circuits, vol. 54, no. 1, pp. 173–185, Jan. 2019.
[14] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss V2: A flexible accelerator for emerging deep neural networks on mobile devices,” IEEE J. Emerging Sel. Topics Circuits Syst., vol. 9, no. 2, pp. 292–308, Jun. 2019.
[15] K.-W. Chang and T.-S. Chang, “VSCNN: Convolution neural network accelerator with vector sparsity,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Sapporo, Japan, May 2019, pp. 1–5.
[16] S.-F. Hsiao, K.-C. Chen, C.-C. Lin, H.-J. Chang, and B.-C. Tsai, “Design of a sparsity-aware reconfigurable deep learning accelerator supporting various types of operations,” IEEE J. Emerg. Sel. Topics Circuits Syst., vol. 10, no. 3, pp. 376–387, Sep. 2020.
[17] Z. Yuan et al., “STICKER: An energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm CMOS,” IEEE J. Solid-State Circuits, vol. 55, no. 2, pp. 465–477, Feb. 2020.
[18] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. J. Lee, “A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 27, no. 8, pp. 1861–1873, Aug. 2019.
[19] Z. Wang, K. Xu, S. Wu, L. Liu, L. Liu, and D. Wang, “Sparse-YOLO: hardware/software co-design of an FPGA accelerator for YOLOv2,” IEEE Access, vol. 8, pp. 116569–116585, 2020.
[20] B.-G. Han, J.-G. Lee, K.-T. Lim, and D.-H. Choi, “Design of a scalable and fast YOLO for edge-computing devices,” Sensors, vol. 20, no. 23, p. 6779, Nov. 2020.
M. Alwani, H. Chen, M. Ferdman, and P. Milder, “Fused-layer CNN accelerators,” in Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Taipei, Taiwan, Oct. 2016, pp. 1–12.

C.-H. Lin et al., “A 3.4-to-13.3 TOPS/W 3.6 TOPS dual-core deep-learning accelerator for versatile AI applications in 7 nm 5G smartphone SoC,” in IEEE ISSCC Dig. Tech. Papers, San Francisco, CA, USA, 2020, pp. J2–134–136.

J. Lee, J. Lee, and H.-J. Yoo, “SRNPU: An energy-efficient CNN-based super-resolution processor with tile-based selective super-resolution in mobile devices,” IEEE J. Emerg. Sel. Topics Circuits Syst., vol. 10, no. 3, pp. 320–334, Sep. 2020.

T. Jin and S. Hong, “Split-CNN: Splitting window-based operations in convolutional neural networks for memory system optimization,” in Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst., Apr. 2019, pp. 835–847.

G. Li, F. Li, T. Zhao, and J. Cheng, “Block convolution: Towards memory-efficient inference of large-scale CNNs on FPGA,” in Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE), 2018, pp. 1163–1166.

K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.

I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, “Designing network design spaces,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 10428–10436.

A. Gordon et al., “MorphNet: Fast & simple resource-constrained structure learning of deep networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Oct. 2018, pp. 1586–1595.

Y. Wang et al., “Pruning from scratch,” in Proc. AAAI Conf. Artif. Intell., Apr. 2020, vol. 34, no. 7, pp. 12273–12280.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes (VOC) Challenge,” Int. J. Comput. Vis., vol. 88, no. 2, pp. 303–338, Sep. 2010.

C.-C. Tsai et al., “The 2020 embedded deep learning object detection model compression competition for traffic in Asian countries,” in IEEE Int. Conf. Multimedia Expo Workshops (ICMEW), London, U.K., 2020, pp. 1–6.

L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” 2017, arXiv:1706.05587.

M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes (VOC) Challenge: A retrospective,” Int. J. Comput. Vis., vol. 111, no. 1, pp. 98–136, Jan. 2014.

K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

O. Russakovsky et al., “ImageNet large scale visual recognition challenge,” Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, Dec. 2015.

Kuo-Wei Chang received the Ph.D. degree from the National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2020. He is currently working with Novatek, Hsinchu. His research interests include system-on-chip (SoC) design and deep learning.

Hsu-Tung Shih received the B.S. and M.S. degrees from the National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2018, and 2020, respectively. He is currently working with Realtek, Hsinchu. His research interest is deep learning.

Chang et al.: REAL-TIME 1280 × 720 OBJECT DETECTION CHIP WITH 585 MB/s MEMORY TRAFFIC 825

Tian-Sheuan Chang (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from the National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1993, 1995, and 1999, respectively. From 2000 to 2004, he was the Deputy Manager with Global Unichip Corporation, Hsinchu. In 2004, he joined the Department of Electronics Engineering, NCTU [as National Yang Ming Chiao Tung University (NYCU), in 2021], where he is currently a Professor. In 2009, he was a Visiting Scholar with IMEC, Leuven, Belgium. His current research interests include system-on-a-chip design, very large scale integration (VLSI) signal processing, and computer architecture. Dr. Chang was a recipient of the Excellent Young Electrical Engineer from Chinese Institute of Electrical Engineering in 2007 and the Outstanding Young Scholar from Taiwan IC Design Society in 2010. He has been actively involved in many international conferences as an Organizing Committee or a Technical Program Committee Member.

Shang-Hong Tsai received the B.S. and M.S. degrees in electrical engineering from the National Kaohsiung University of Applied Sciences, Kaohsiung, Taiwan, in 2015 and 2017, respectively. He is currently a Principle Engineer with the Taiwan Semiconductor Research Institute (TSRI), Hsinchu. His research interests include very large scale integration (VLSI) design and hardware implementation.

Chih-Chyau Yang (Associate Member, IEEE) received the B.S. degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 1996, and the M.S. degree in electronics engineering from the National Chiao Tung University, Hsinchu, Taiwan, in 1999. He is currently a Principal Engineer with the Taiwan Semiconductor Research Institute (TSRI), Hsinchu. His research interests include very large scale integration (VLSI) design, computer architecture, and platform-based system-on-chip (SoC) design methodologies.

Chien-Ming Wu (Member, IEEE) received the B.S. and M.S. degrees in electronic engineering from the National Yanglin University of Science and Technology, Douliu, Taiwan, in 1997 and 1999, respectively, and the Ph.D. degree from the Graduate School of Engineering Science and Technology, National Yunlin University of Science and Technology, in 2003. He is currently a Researcher Fellow and the Deputy Division Manager with the Taiwan Semiconductor Research Institute (TSRI), Hsinchu, Taiwan. His research interests include very large scale integration (VLSI) design, computer architecture, and platform-based system-on-chip (SoC) design methodologies.

Chun-Ming Huang (Member, IEEE) received the B.S. degree in mathematical science from the National Chengchi University, Taipei, Taiwan, in 1990, and the M.S. and Ph.D. degrees in computer science from the National Tsing-Hua University, Hsinchu, Taiwan, in 1992 and 2005, respectively. He is currently a Research Fellow and the Division Manager with the Taiwan Semiconductor Research Institute (TSRI), Hsinchu. His research interests include very large scale integration (VLSI) design and testing, platform-based system-on-chip (SoC) design methodology, and multimedia communication. Dr. Huang is a member of Phi Tau Phi Scholastic Honor Society.