CNN-DMA: A Predictable and Scalable Direct Memory Access Engine for Convolutional Neural Network with Sliding-window Filtering

Zheng Wang*

Zhuo Wang*

Jian Liao

Chao Chen

zheng.wang@siat.ac.cn
Shenzhen Institute of Advanced
Technology, CAS

Yongkui Yang

Bo Dong

Weiguang Chen

Wenxuan Chen

Ming Lei

Shenzhen Institute of Advanced
Technology, CAS

Weiyu Guo

Rui Chen

Yi Peng

Zhibin Yu

Shenzhen Institute of Advanced
Technology, CAS

ABSTRACT

Memory bandwidth utilization has become the key performance bottleneck for state-of-the-art variants of neural network kernels. Current structures such as depth-wise, point-wise and atrous convolutions have already introduced diverse and discontinuous memory access patterns, which impact efficient activation supply due to more frequent cache misses and consequently high-penalty DRAM pre-charging. To handle this, GPU achieves efficient parallelization with sophisticated optimization of CUDA program to reduce memory footprints, which demands high engineering efforts. In this work, we in contrast propose a programmable direct memory access engine for convolutional neural networks (CNN-DMA) supporting a fast supply of activation for independent and scalable computing units. The CNN-DMA favours a predictable activation streaming approach which completely avoids penalties by bus contention, cache misses and less carefully designed low-level programs. Furthermore, we enhance the baseline DMA with the capability of out-of-order data supply to filter out unique sliding-windows to boost the performance of the computing infrastructure. Experiments on state-of-the-art neural networks show that CNN-DMA achieves optimal DRAM access efficiency for point-wise convolution layers, while reduces 30% to 70% rounds of computation with sliding-window filtering.

CCS CONCEPTS

• **Hardware** → **Application specific integrated circuits**; *Dynamic memory*; • **Computer systems organization** → **Neural networks**; **Data flow architectures**.

*Both authors contributed equally to this research. Zheng Wang is the corresponding author (zheng.wang@siat.ac.cn).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8393-6/21/06...\$15.00

<https://doi.org/10.1145/3453688.3461496>

KEYWORDS

Direct memory access, machine learning accelerator, sliding-window filtering

ACM Reference Format:

Zheng Wang, Zhuo Wang, Jian Liao, Chao Chen, Yongkui Yang, Bo Dong, Weiguang Chen, Wenxuan Chen, Ming Lei, Weiyu Guo, Rui Chen, Yi Peng, and Zhibin Yu. 2021. CNN-DMA: A Predictable and Scalable Direct Memory Access Engine for Convolutional Neural Network with Sliding-window Filtering. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22–25, 2021, Virtual Event, USA*. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3453688.3461496>

1 INTRODUCTION

The recent advancements in machine learning, especially neural networks (NN), have demonstrated their success in a wide range of application domains, such as computer vision and natural language processing. To accelerate the massive amount of computation in neural networks, domain-specific architectures and circuits have been designed[3][4], which exploits multiple levels of parallelism to primarily improve energy efficiency. However, as degrees of parallelism have been exhaustively investigated, a huge amount of external memory access has rapidly become the de facto bottleneck to further improve energy efficiency. The root cause is that state-of-the-art NN employs 10MB~200MB coefficients and generates a comparable size of activations, which are both ways too costly to be solely stored in on-chip SRAMs. Before process-in-memory (PIM) technology comes to a well-developed and robust stage, DRAM will keep playing a key role for housing coefficients and activations, therefore unavoidably increases system latency and consumes a major percentage of system power. Degradation in energy efficiency has been foreseen for the designs with high degrees of computational parallelism but fails to efficiently exploit memory bandwidth.

For academic designs, customized dataflow has been proposed to reduce DRAM footprint for selective network layers such as shortcut[1]. However, optimizations for specific algorithmic kernels usually fails to be generalized to the rest. For instance, kernels such as compact convolutions [6] with multiple dilation levels [2], increases irregularity in memory access pattern but still demands generic data supply techniques with fewer DRAM accesses. In contrast, industrial designs such as GPU is built with intrinsic flexibility

provided by low-level CUDA APIs. Optimizations towards GPU memory access are possible but usually involve large engineering complexity [5][8], which can be partially attributed to the memory subsystem of modern GPUs.

Multi-level caches have been widely adopted in NN accelerators and GPUs. The intuition is that increased memory hierarchy can reduce the amount of remote DRAM access when considering data locality. However, hundreds of parallel executing threads can easily compete for data in the l2-cache which leads to bus contention and increased unpredictability of system latency. For general-purpose computing, such penalty is either tolerated or can be partially released by manipulating CUDA APIs. However, for the majority of NN kernels, we claim that the costly l2-storage can be completely discarded to reduce system unpredictability.

In this work, we propose CNN-DMA, a direct memory access engine for the convolutional neural network to replace the conventional l2-storage and supplies activations and weights directly to l1-buffers associated with individual computing threads. By understanding patterns of various NN kernels, CNN-DMA adopts a *Fetch&Jump* data acquisition mechanism to assist individual l1-buffers to precisely and independently catch required data in the assembled data-stream. Predictability is achieved since cache miss penalty and bus contention is both avoided in CNN-DMA.

Furthermore, we enhance the baseline CNN-DMA with the capability of out-of-order allocation of l1-buffers. The idea is inspired by the fact that most l1-buffers, holding adjacent sliding-windows in CNN, shows high similarity in data contents which can be saved for repeat execution. CNN-DMA automatically performs on-the-fly buffer content analysis and assigns available l1-buffers only to the unique sliding-windows, while reconstructing correct output activations during post-processing. Consequently, the CNN-DMA with sliding-window filtering achieves predictable memory access while greatly reduces the computation time and energy in the array of processing elements (PEs).

The work is organized as follows. Section 2 presents the design principle of baseline CNN-DMA. Section 3 introduces the enhancement with sliding-window filtering. Section 4 details the architecture of CNN-DMA and its integration in a neural network computing engine. Section 5 illustrates the experimental methodology, achieved performance and physical estimates. Section 6 concludes the work.

2 PREDICTABLE CNN-DMA ENGINE

The core design principle of the CNN-DMA lies in its predictable activation fetching methodology which allows the engine to efficiently fetch data from DRAM. This section explains the motivation, details the procedure of proposed data fetching operations and introduces an analytical model for fast estimation of DRAM access efficiency for CNN-DMA.

2.1 Motivation

Convolution and pooling kernels in the neural network can be both visualized as a 3-D kernel sweeping through the input feature map (Ifmap) to generate the output feature map (Ofmap). NN accelerators and GPUs increase computational parallelism by multi-threaded processing of connected pixels in Ofmap, which is illustrated as one single slice in Fig. 1a). Each pixel in a particular slice

has a perception field in the Ifmap named as *sliding-window* of activations, which convolves with convolution kernel or computing maximal, minimal and average values for pooling kernel. The sliding-windows of elements in a slice strongly overlap in the Ifmap, which forms a region as shown in Fig. 1b). An l2-buffer is usually adopted to first load the complete region and later accessed by individual hardware threads. However, most activations in a region belong to several sliding-windows, which needs to be repeatedly accessed among threads and therefore leads to l2-buffer contention. The problem gets exaggerated for regions with zero-padding where several threads need to fill zero while others access valid data in the region. Such branching situation incurs extra overheads to the single-instruction-multiple-threads (SIMT) execution model of GPUs.

2.2 Fetch&Jump activation acquisition

We advocate supplying sliding-windows from DRAM to l1-buffers directly through DMA. However, in contrast to conventional DMA which streams desired data to a specific master based on address and length information, the proposed CNN-DMA has to be extended with two features. First, a single CNN-DMA activation stream can target *tens to hundreds* of master threads, which on-the-fly catch individually and differently desired data elements. Second, the DMA should dynamically handle the zero-padding issues to form a continuous Ifmap region instead of a continuous physical data section.

Illustrated in Fig. 1c), after determining region in Ifmap, the CNN-DMA subdivides the region into several streams based on the rows of algorithmic kernels. Each stream starts from the top-left of a specific stream and ends at top-right of the same stream. A stream can easily cover multiple rows in the region for a large number of parallel threads thus the top-right position can shift to next rows. Padding is inserted on the stream borders according to the position of the targeting region in the entire Ifmap. The finally assembled streams are accessible by parallel l1-buffers.

Show in Fig. 1d), to catch desired data each l1-buffer is equipped with a fetch pointer and low-cost jumping logic. The pointer is programmed based on the kernel structure, such as width, height, dilation level, stride and padding. Pointed data in the assembled stream is fetched into the specific l1-buffer. Afterwards, the pointer is dynamically updated to point at the position of next desired data in the same stream. When the current stream is finished, the pointer resets to the first position of the next stream. Therefore after all the streams, all l1-buffers are filled with desired data and incurs no contention or penalty.

2.3 Analytical model

The proposed data acquisition approach is predictable in the amount of DRAM access. We establish an analytical model to estimate the amount of Ifmap transferred from DRAM to l1-buffers, present in the following.

Figure 1: Illustration of segmenting Ifmap region into activation streams and Fetch & Jump activation acquisition methodology

$$\begin{aligned}
 \text{Sets} &= \text{floor}(Wo * Ho / \text{Threads}) \\
 \text{Remains} &= \text{mod}(Wo * Ho, \text{Threads}) \\
 \text{Pad} &= Sw * Wo - Wi + Fw - Sw \\
 \text{Amount} &= \text{Sets} * Fh * Ci * (\text{Threads} * Sw \\
 &\quad + 2 * (Fw - Sw) - \text{Pad}) \\
 &\quad + Fh * Ci * (\text{Remains} * Sw \\
 &\quad + 2 * (Fw - 2 * Sw) - \text{Pad}) \\
 &\quad - \text{Pad} * Wi * Ci
 \end{aligned} \tag{1}$$

Wo and Ho represent the Ofmap width and height. Wi is the width of Ifmap. Fw and Fh are width and height of the kernel. Ci denotes the Ifmap channel number. Sw is the stride. Threads denotes the computing thread number. Sets represent the iterations to finish processing Ofmap; Remains is the result of modulo operation; Pad is the number of zero-paddings on both sides; Amount gives the total amount of data transfer. The analytical model applies to standard, depth-wise, point-wise, atrous convolution and various pooling kernels. Cycle-accurate simulation across various kernels indicates that the mean error of the model is less than 0.4%.

Figure 2: Analytical model for computation and DRAM access with different kernels

Using the model, it is interesting to investigate the DRAM access quantitatively of proposed CNN-DMA. We list the size of Ifmap and

associated total amount of MACs for various kernel configurations and the number of loaded activations from our model in Fig. 2. It is shown that as of kernel dimension increases, the number of MACs significantly increases while the size of Ifmap remains. Ideally, the number of loaded activations from DRAM should equal the Ifmap size. Our DMA incurs 0%, 50%, and 400% overheads for 1×1 , 3×3 and 7×7 kernels, which achieves the operation intensity (number of MACs per loaded activation) of 1, 1.5 and 3.3 respectively.

3 CNN-DMA WITH SLIDING-WINDOW FILTERING

Conventional DMA is designed to initiate a fast data transfer. However, data transfer is usually followed by costly data preprocessing in modern SoCs, which repeatedly walks through the cached data. The core idea of this section is to build preprocessing as a parallel procedure to data transfer, therefore saving another run of data access.

In the neural network, similarity exists among adjacent sliding-windows due to temporal or spatial localities in the Ifmaps. Such similarity is reused to increase power efficiency in [9][10]. However, reusable features are currently integrated into the PE array, which may reduce dynamic power during computation but increase design complexity and show less gain in performance. We, on the other hand, propose to integrate sliding-window filtering in the CNN-DMA during the procedure of data acquisition.

The baseline CNN-DMA in section 2 distributes activations into l1-buffers in the order that adjacent l1-buffers hold adjacent sliding-windows. However, for the similar sliding window which can be skipped for computation, its l1-buffers can be left to fill other unique sliding windows in the next stream (Fig. 1d)). Consequently, the resulted l1-buffers after filtering contains out-of-order sliding-windows.

Fig. 3 visualizes out-of-order l1-buffer allocation using sliding-window filtering, where each box represents a thread buffer (TB). The number of TBs is set to 25 for demonstration. TBs are initialized to be empty with blank colour. After the 1st stream, all 25 TBs are filled with sliding-windows (denoted using blue colour). Filtering is conducted in parallel to eliminate similar TBs, which changes the colour to blank. Afterwards, the 2nd stream fills unused TBs

to brown colour. The process iterates until all TBs are filled with unique sliding-windows, before the start of computation.

Figure 3: Out-of-order level-1 buffer allocation with sliding-window filtering

3.1 Similarity determination

Fig. 4 details the out-of-order threads allocation. In Fig. 4(a), as the data from activation stream are fetched into thread buffers, we extract the maximum, minimum, and sum of data on the fly. When all 25 sliding-windows are filled, we compare the extracted features of adjacent thread buffers. If the difference between them is less than a preprogrammed threshold value, we conclude that two sliding-windows are similar, hence mark the previous thread buffer data as valid, and the latter one as invalid. We encode the validity of each thread in a table (i.e. *thread readiness truth table*). Thresholds are programmed through network instructions, which can be tuned during execution.

3.2 Mapping

The sliding-window filtering increases the complexity of thread management caused by out-of-order buffers. To alleviate such issue, a *logi-physic-thread-mapping* table is introduced to bookkeep the dynamic mapping between logical and physical threads. The address of the mapping table indicates the logical thread number for the current stream and the addressed data identifies the mapped physical thread number. For instance, as illustrated in Fig. 4(b), 13 TBs after the 1st stream are marked as unused, which are open to being filled by the following streams. After filtering the 2nd and 3rd streams, we have 6 open TBs and 2 open TBs. The table is updated until all TBs are filled with filtered sliding-windows.

3.3 Results reconstruction

When all TBs are filled, the computation begins in the PE array. The baseline DMA employs in-order threads therefore results from PEs can be committed sequentially. On the contrary, the DMA with out-of-order TBs is subject to commit in order Ofmaps from out-of-order PE outputs which leads to the procedure of Ofmap reconstruction. An *Ofmap reconstruction table* is introduced to record the modes of each committed result from the PE array and guide the DMA

for reconstruction. The *ofmap reconstruction table* is synchronized with *logi-physic-thread-mapping* table dynamically.

Two modes of reconstruction are explained using an example. In Fig. 4(c), The 3rd thread after 1st stream contains filtered activations and $16'b\{8'b0,0000_0011\}$ is written to the *ofmap reconstruction table*. The 8 LSBs represent the physical thread number, where the 4th, 5th, and 6th physical threads are invalid due to their similarity to the 3rd thread. We encode such situation as $16'b\{8'b1,0000_0011\}$, where the 8 LSBs represent the amount of similar threads. During reconstruction, each field of 16 bits is read from the table. If the 8 MSBs equal to zero, the result of the thread number stored in the 8 LSBs is written to DRAM. Otherwise, if the 8 MSBs equals to 1, the result of the last writeback is repeated where the 8 LSBs are assigned as the number of repeated writebacks.

4 ARCHITECTURE

4.1 Baseline NN accelerator

The baseline computing infrastructure, named as *Multiple-Weights-Multiple-Threads* (MWMT) array, exploits an output stationary dataflow where the partial sums are accumulated in a two-dimensional PEs. The PEs are organized in multiple threads as present in the pink aura of Fig. 5. The PEs within the same thread compute in parallel multiple output channels sharing the same Ifmap sliding-window. Therefore, multiple threads can process multiple Ifmap sliding-windows simultaneously. Each thread has a unique thread buffer (TB) for loading and sharing the activations. Network weights are preloaded into the weight buffers (WB) and shared among PEs which computes the same channel in the Ofmap. Outputs are first written to the commit buffer (CB) for splicing segmented activations before committing to the external DRAM. The MWMT array facilitates synchronized step-locking across threads, which reduces the design cost of inter-PE communication and management.

4.2 Input activation acquisition

The CNN-DMA is designed under the guidelines in Section 2 and 3 and integrated into the baseline accelerator, which implements an AXI-master in the system-on-chip. Same as conventional DMA, the CPU initializes a request and will be interrupted once NN accelerator completes its execution.

The CNN-DMA consists of several modules to implement predictable data-streaming with activation filtering. Explicitly, The NN access pattern generator (APG) initializes DRAM transfers by translating the network properties into segmented DRAM read accesses as shown in Fig. 1d). The raw loaded data segments from DRAM is assembled with appropriate zero paddings to form a continuous data-stream, which is simultaneously accessible from multiple Fetch & Jump (F&J) pointers. The F&J pointers are preprogrammed by the APG to fetch individual activation values for each thread according to the kernel properties. After catch current values, the pointers are automatically updated to prepare to catch upcoming activations. Besides activations, weights are also streamed through the assembler and fetched into WBs. The weight streams are not caught using F&J features due to their relatively static properties, which are pre-handled during NN compilation.

Figure 4: Operating flow of similarity-based sliding-window filtering: a) similarity evaluation of adjacent thread buffers. b) dynamic mapping between logical threads and physical threads. c) reconstruction of in-order output feature maps.

4.3 Sliding-window filtering

The sliding-window filtering is achieved through on-the-fly feature extraction of fetched data. The extracted features are temporarily stored into the feature registers associated with each thread. For instance, in this work maximal, minimal and cumulated values can be acquired for judging the similarity of adjacent sliding-windows. The dynamic thread allocation module is used to mark TB and thread as *ready* for filtered sliding-window. The decision threshold of readiness is pre-programmed and varies across network layers. Further filtering principles can be also introduced and implemented. The feature extraction, comparison and thread allocation are all performed in parallel to TB loading, which incurs no extra timing overhead.

With the sliding-window filtering turns on, one activation stream normally marks only several threads to be ready. To minimize control flow overheads, the MWMT array only starts execution when all threads are ready. Therefore, subsequent activation streams are demanded to fill the rest TBs. To achieve such iterated streaming-filtering mechanism, the APG is dynamically updated with thread readiness information, prepares corresponding memory transfer and directs data-stream assembler and F&J pointers to acquire updated activations. The key feature of the thread controller is to maintain the mapping table between logical and physical threads.

4.4 Output activation reconstruction

As mentioned in section 3, sliding-window filtering introduces out-of-order computation in MWMT array where physically adjacent threads process logically distant sliding-windows. Consequently,

output activations have to be reconstructed into logically adjacent values before storing to DRAM. To achieve this, the commit buffer is first filled with out-of-order data and then sends those by passing through the in-order reconstruct (IOR) module. The IOR decodes both logi-physi-mapping table and reconstruction table to append repeated activations to the activation stream where necessary. The augmented streams are finally transferred from CNN-DMA to AXI interface.

4.5 Programmability and scalability

The CNN-DMA provides programming registers to achieve desired functionalities. Table 1 lists selected registers and explanations. The DMA can be delivered as a stand-alone AXI master, or integrated into state-of-the-art NN accelerators for fast data distribution.

In terms of scalability, the MWMT array does not put any constraint on the allowed amount of sliding-windows to be computed in parallel. However, conventional data supply technique, such as employing l2-cache for storing shared regional activations, bears heavy burdens on the latency of l1-cache loading when an increased amount of threads demand data simultaneously and therefore causes contention. In contrast, the proposed approach uses CNN-DMA to assemble data-streams for parallel threads to fetch individually. One limitation for scalability is the maximal fanout size of multiple fetch pointers, which impacts operating frequency. However, it can be confidently claimed that at least 100 threads can fetch data concurrently under advanced technology nodes.

Figure 5: Architecture of proposed accelerator with CNN-DMA

Table 1: Selected programming registers for CNN-DMA

Register naming	Category	Explanation
Ifmap (H,W,C)	Network and kernel structures	Height, width, channel
Ofmap (H,W,C)		
Kernel (H,W,C)		
Dilation level		
Zero-padding (L,R,T)		
Stride	Sliding window filtering	Amount of holes in kernel
Filter enable		Paddings: Left, right, top
Strategy		Step size of sliding-window
Threshold		'1' for enable filtering
Threads amount	Micro-archs.	Max, min, sum of window
PEs per thread		Difference between windows
Batch size		Amount of parallel threads
		Amount of PEs per thread
		Concurrently proc. Ifmaps

5 EXPERIMENTAL RESULTS

The CNN-DMA with sliding-window filtering and the baseline NN engine is completely designed in Verilog. Logic synthesis at 400MHz under 40nm SMIC technology is performed for physical estimations. The architecture prototype is synthesized onto Kintex-7 FPGA to evaluate the performance of neural networks with 50MB+ weights such as AlexNet. Opalkelly[7] IPs are used for interfacing FPGA with host machine as testing infrastructure. A NN software kit, including model compilation, quantization and deployment is developed and achieves compatibility with Keras and Pytorch frameworks.

5.1 Performance evaluation

The MWMT array with 256 to 2,048 PEs (8 to 64 threads and 32 PEs per thread) is evaluated with 100Mhz frequency on Kintex-7 FPGA. As shown in Fig. 6. As number of threads increases from 8 to 64, the execution time for individual layers roughly reduces 75% which indicates the scalability of proposed CNN-DMA. However, increased parallelism also enlarges the mapped region in Ifmap therefore increases the timing overhead for activation streaming.

5.2 DRAM access benchmarking

Fetches activations are measured from DRAM and benchmarked with the size of Ifmap (lower bound of activation supply) and MACs (upper bound of activation supply) to verify the analytical model of CNN-DMA in Section 2. We choose MobileNet for evaluation since it consists of regular, depth-wise and point-wise convolution kernels. Ideally, the fetched activations should approach the size of Ifmaps. As shown in Fig. 7, all point-wise layers have an identical amount of fetched data to Ifmaps which achieves optimal in fetching activations. On the contrary, depth-wise layers fetch the most amount of activations, caused by more frequent thread buffer refilling due to update of weights. Regular convolution lies between two compact layers in terms of DRAM access efficiency.

5.3 Sliding-window filtering

We evaluate the performance of NN-engine equipped with CNN-DMA running classical NN models on MNIST and ImageNet datasets. The decision criterion of similarity is based on the summation of maximal, minimal and accumulated values in the sliding-window. In case the difference of summation between TBs are larger than zero, they are treated as both valid TBs. Other threshold values which are larger than zero will be explored in the future work.

Table 2 shows the total rounds of computation to complete a specific network layer in LeNet and AlexNet, where each round processes 2,048 elements in the Ofmap concurrently. It is observed that the default CNN-DMA without sliding-window filtering requires 13 rounds to process LeNet Conv-1 layer. On the contrary, CNN-DMA with filtering reduces the number of rounds to four, which offers a 69% saving in both execution time and computation power. For AlexNet, saving of rounds with filtering is less significant but still larger than 30%. The reason for the difference in saving is that images in MNIST have a relatively monotonous background (black and white) while the ones in ImageNet are abundant in colours therefore exhibit less similarity among sliding-windows.

Table 2: NN-engine performance for with CNN-DMA

Layer index	Ofmap Size	NN-engine w. CNN-DMA		OPS saving
		w.o. filter	with filter	
LeNet on MNIST dataset				
Conv-1	$28 \times 28 \times 6$	13	4	69%
Conv-2	$10 \times 10 \times 16$	2	1	50%
Total	--	15	5	67%
AlexNet on ImageNet dataset				
Conv-1	$55 \times 55 \times 96$	48	30	37%
Conv-2	$27 \times 27 \times 256$	12	8	32%
Conv-3	$13 \times 13 \times 384$	3	2	33%
Conv-4	$13 \times 13 \times 384$	3	2	33%
Conv-5	$13 \times 13 \times 256$	3	2	33%
Total	--	69	46	33%

5.4 Physical overheads

Physical estimates through Synopsys design compiler is present in Table 3, which include both MWMT array and CNN-DMA engine. The baseline architecture without sliding-window filtering achieves

Figure 6: Measured performance of MWMt prototypes on Kintex-7 FPGA synthesized at 100MHz

Figure 7: Statistics of computation and measured DRAM access

a maximal frequency of 450MHz. In contrast, the involvement of sliding-window filtering prolongs the critical path and results in a 400MHz design. The area overhead is introduced mainly due to 4KB SRAM for reconstruction table and associated combinatorial logic, while the other tables are implemented with registers. 8% overhead in power consumption is estimated. Finally, the design with filtering has less peak physical throughput compared to baseline design due to lowering of maximal frequency, but on the other hand, achieves dynamic boosting in performance due to sliding-window filtering. A rough estimate of $1\times \sim 3\times$ boosting can be reasoned from Table 2 but the precise effect varies with activations of neural network.

Table 3: Physical properties of NN-engine including CNN-DMA

Technology	40nm SMIC CMOS	
	w.o. filter	with filter
Synthesis frequency (DC)	450MHz	400MHz
Area estimate (DC)	7.96mm ²	8.25mm ²
Power estimate (DC)	452.5mW	489.2mW
Total SRAM size	1024KB	1028KB
Number of PEs	2048	2048
Throughput(GOPs)	1, 843(Peak)	1, 638(Peak)) + turbo filtering

6 CONCLUSION

Level-2 buffers are widely adopted in the state-of-the-art GPUs and NN accelerators to reduce the amount of DRAM access. However as level of parallelism grows, bus contention and cache miss happens more frequently which degrade the system performance and power efficiency. In this work, we present the design principle of a direct memory access engine for neural networks, which delivers sliding-windows into thread buffers in parallel to avoid the unpredictability of l2-buffer. Furthermore, the CNN-DMA is extended with the feature of slide-window filtering, which saves rounds of execution based on dynamic similarity analysis. A prototyping computing array with CNN-DMA is built which verifies the DRAM access efficiency and filtering capability.

ACKNOWLEDGMENTS

This work was supported by Key-Area Research and Development Program of Guangdong Province (Grant No. 2019B010155003), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2020B1515120044), National Natural Science and Foundation of China (NSFC 61902355).

REFERENCES

- [1] Arash Azizimazreah and Lizhong Chen. 2019. Shortcut mining: Exploiting cross-layer shortcut reuse in dcnn accelerators. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 94–105.
- [2] Liang Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking Dilated Convolution for Semantic Image Segmentation. In *Google Inc.*
- [3] Yunji Chen et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.
- [4] Yu-Hsin Chen et al. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.
- [5] NVIDIA Corporation. 2020. CUDA C++ Programming Guide PG-02829-001-v11.0. In <https://docs.nvidia.com/cuda/pdf/CUDA-C-Programming-Guide.pdf>. NVIDIA Corporation, 340–346.
- [6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
- [7] Opal Kelly. 2013–2016. <https://opalkelly.com/>.
- [8] Nuno Neves, Pedro Tomás, and Nuno Roma. 2018. Stream data prefetcher for the GPU memory interface. *The Journal of Supercomputing* 74, 6 (2018), 2314–2328.
- [9] Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. Computation reuse in DNNs by exploiting input similarity. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 57–68.
- [10] Zhe Yuan et al. 2020. 14.2 A 65nm 24.7 μ J/Frame 12.3 mW Activation-Similarity-Aware Convolutional Neural Network Video Processor Using Hybrid Precision, Inter-Frame Data Reuse and Mixed-Bit-Width Difference-Frame Data Codec. In *2020 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 232–234.