Exploration of Memory Access Optimization for FPGA-based 3D CNN Accelerator

Teng Tian, Xi Jin*, Letian Zhao, Xiaotian Wang, Jie Wang and Wei Wu
Key Laboratory of Strongly-Coupled Quantum Matter Physics, Chinese Academy of Sciences
Institute of Microelectronics, School of Physical Sciences, University of Science and Technology of China. Hefei, Anhui, China
{tianteng; zhaolt; wxtdsg; wangj1e; wuw1993}@mail.ustc.edu.cn, *jinxi@ustc.edu.cn

***Abstract*—Three-dimensional convolutional networks (3D CNNs) are used efficiently in various video recognition applications. Compared to traditional 2D CNNs, extra temporal dimension causes 3D CNNs more computationally intensive and to have a larger memory footprint. Therefore, the memory optimization is extremely crucial in this case. This paper presents a design space exploration of memory access optimization for FPGA-based 3D CNN accelerator. We present a non-overlapping data tiling method for contiguous off-chip memory access and explore on-chip data reuse opportunity by leveraging different loop ordering strategies. We propose a hardware architecture design which can flexibly support different loop ordering strategies for each 3D CNN layer. With the help of hardware/software co-design, we can provide the optimal configuration toward an energy-efficient and high-performance accelerator design. According to the experiments on AlexNet, VGG16, and C3D, our optimal model reduces up to 84% DRAM accesses and 55% energy consumption on C3D compared to a baseline model, and demonstrates state-of-the-art performance compared to prior FPGA implementations.**

Keywords—3D CNN, data tiling, loop ordering, energy-efficient

I. Introduction

In recent years, *convolutional neural networks* (CNN) are attracting much attention as they have achieved near-human accuracy on numerous tasks, e.g., image classification [1], object detection [2], and face recognition [3]. In the coming era of 5G communication, video-based applications will be further popularized and revolutionized, and many of them rely on video understanding, such as human action recognition [4], video classification [5], and medical image diagnosis [6]. The state-of-the-art accuracy on video recognition is achieved using *three-dimensional convolutional neural networks* (3D CNNs).

Compared to 2D CNNs used for image recognition, 3D CNNs with spatiotemporal convolutional kernels compute features from contiguous frames, which thereby can capture motion information on the temporal dimension. Meanwhile, the input features, weights and *partial sums* (*psums*) of 3D convolution consume more memory space than 2D convolution, making it more difficult to fit them in on-chip memory. Data tiling and loop optimization strategies for 2D CNNs have been well-studied in [7]. Since more data reuse opportunities can be captured on the temporal dimension in 3D CNNs, 2D CNN accelerators without temporal data reuse are inefficient when evaluating 3D CNNs. Morph [8] tiles input data on all dimensions in 3D convolution and flexibly supports all possible loop orders to accelerate 3D CNNs in ASIC. However, tiling data on spatial and temporal dimensions in 3D convolution introduces a higher degree of data replication than that in 2D convolution. Moreover, non-contiguous and unaligned DRAM access increase the memory overheads with improper tile sizes and loop orders. Fixing a specific data type (e.g., input features) statically in on-chip memory [9] is also a sub-optimal design philosophy for 3D CNN accelerator, for the memory requirements of features and weights vary significantly across convolutional layers in 3D CNNs. Besides, the on-chip energy contributes a higher proportion to the total energy consumption of evaluating 3D CNNs since the data reuse in 3D CNN is much higher. Consequently, the memory optimization is extremely crucial for an energy-efficient and high-performance 3D CNN accelerator design.

Owing to the advantages of flexibility and energy efficiency, FPGAs have become a particularly attractive option to accelerate CNN inference [7, 10] and training [11, 12]. Here we focus on inference. Abundant FPGA-based CNN accelerators target 2D CNNs, while the attempts made to accelerating 3D CNNs are quite few. F-C3D [10] performs 3D convolution by blocking data on temporal or spatial dimensions. This design philosophy suffers from data overlap and caching all data on channel dimension which is typically a large number in later layers. The design in [13] extends the Winograd algorithm to adapt to 3D convolutions. However, the transformation matrices in Winograd algorithm are closely related to the size of convolutional kernels. Hence their design is not suitable for CNN models with multi-sized convolutional kernels. Mapping 3D convolution to matrix multiplication is implemented in [14]. Their accelerator achieves higher performance density on 2D CNN rather than 3D CNN, for a higher degree of data replication is introduced in 3D convolution.

In this paper, we present a design space exploration of memory access optimization for an energy-efficient and high-performance 3D CNN accelerator design based on FPGA. The main contributions of this work are summarized as follows:

- We present a non-overlapping data tiling method between on-chip and off-chip memory, which introduces contiguous off-chip memory access and eliminates the overhead of overlapping data. Then we exploit on-chip data reuse and data parallelism by leveraging different loop ordering strategies.
- We propose a hardware architecture design that takes advantages of spatiotemporal data locality and can flexibly support different loop ordering strategies for each 3D CNN layer. We further co-design a software infrastructure to find the optimal configuration for energy or performance under different hardware resource constraints.

Sponsored by Huawei Innovation Research Program.

- We evaluate our design on AlexNet, VGG16, and C3D. Experimental results show that our optimal model reduces up to 84% DRAM accesses and 55% energy consumption on C3D compared to a baseline model, and achieves state-of-the-art performance on Xilinx VC707 board compared to prior FPGA implementations.

The rest of this paper is organized as follows: Section II describes data tiling and loop ordering strategies for 3D CNN. Section III presents the hardware architecture design. Section IV describes the software optimization framework. Section V evaluates our design, and Section VI concludes this paper.

II. Design Methodology

A. Data Tiling

Fig. 1 depicts a typical 3D convolution with valid padding. An input feature of spatial size $H \times W$, temporal dimension D and C channels, is 3D convolved channel by channel with M filters (containing weights) of spatial size $R \times S$, temporal dimension T and C channels. Assume the strides are (1, 1, 1) on spatial and temporal dimensions, 3D convolution produces M *psums* of spatial size $(H-R+1) \times (W-S+1)$, temporal dimension $(D-T+1)$ and C channels. Then the *psums* are accumulated on channels to produce output features with M channels. 2D convolution is a special case of 3D convolution with $D = T = 1$.

Accelerators typically perform CNNs layer by layer and access data of each layer from expensive off-chip DRAM to low-cost on-chip SRAM. Because the input features and weights may not fit in on-chip memory, the accelerator has to tile the data to perform 3D convolution. However, there are five dimensions (*H, W, D, C,* and *M*) that need to be considered when tiling data. Note that R, S and T are generally small values, thereby not considered for tiling. Prior works [8, 10] attempted to tile data on spatial and temporal dimensions and broke data into tiles of size $H_t \times W_t \times D_t \times C_t \times M_t$. Each convolutional filter slides across H, W, and D dimensions of the input feature map, hence adjacent input tiles on these dimensions will overlap if the convolution strides are smaller than tile dimensions. Assume H=W, R=S=T=K, H_t=W_t, the memory overhead of 2D and 3D convolution can be calculated by (1) and (2), respectively:

$$OH_{2D} = 1 + \frac{2(K-1)H(N_H-1)}{HW} = 1 + \frac{2(K-1)(N_H-1)}{W} \tag{1}$$

$$OH_{3D} = 1 + \frac{2(K-1)HD_t(N_H-1)+(K-1)HW(N_D-1)}{HWD}$$
$$= 1 + \frac{(K-1)[2D_t(N_H-1)+W(N_D-1)]}{WD} \tag{2}$$

where N_H and N_D are the numbers of tiles on H and D dimensions. Note that the memory overhead of 3D convolution (OH_{3D}) is larger than 2D convolution (OH_{2D}) when D_t is not equal to D. As shown in Fig. 2, the memory overhead is proportional to the number of tiles and has a greater effect on 3D convolution. Furthermore, tiling data on H and W dimensions will introduce non-contiguous DRAM access and result in a large number of memory transfers with short burst lengths. Moreover, improper tile size on spatial and temporal dimensions can cause unaligned DRAM access.

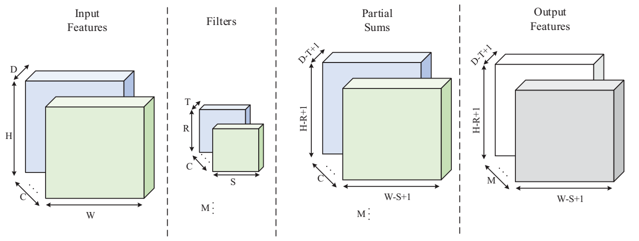

Fig. 1. Hyperparameters in 3D convolution. Assumes the strides are (1, 1, 1) on spatial and temporal dimensions, and valid padding is performed.

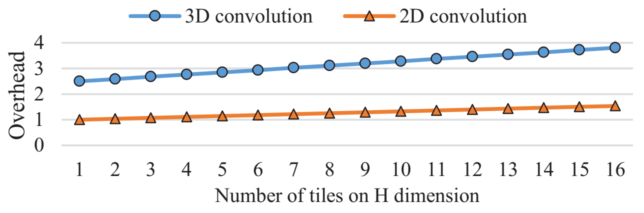

Fig. 2. The memory overhead of 2D and 3D CNNs when tiling inputs on the spatial dimension. Assumes H=W=112, D=16, R=S=T=3, H_t=W_t, D_t=3, and the stride is 1.

TABLE I. Partial Hyperparameters of C3D

layers	1	2	3	4	5	6	7	8
H(W)	112	56	28	28	14	14	7	7
D	16	16	8	8	4	4	2	2
C	3	64	128	256	256	512	512	512
M	64	128	256	256	512	512	512	512

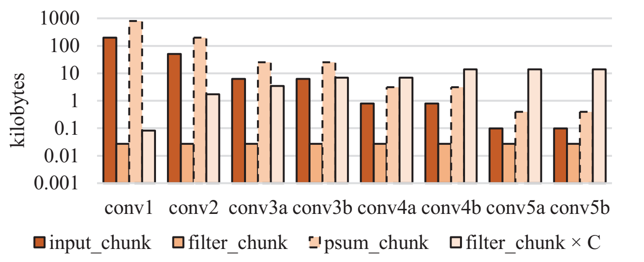

Fig. 3. Data chunk size of C3D when adopting non-overlapping strategy across all layers. The input size of the first layer is 3×16×112×112.

On the contrary, there is no overlapped data on C and M dimensions. Table I lists the hyperparameters of *H, W, D, C,* and *M* across convolutional layers of C3D [5]. Note that *H, W,* and *D* decrease in a geometric ratio in later layers, while the trend of C and M is opposite. Grouping $H \times W \times D$ data as minimum tiling size (*chunk*) is an optimal solution in later convolutional layers, due to contiguous off-chip memory access and no overlapped data between *chunks*. However, what about the early layers?

Fig. 3 illustrates the *chunk* size of inputs, weights, and *psums* of C3D. We assume the inputs and weights are quantized to 8-bit as this achieves better performance without accuracy drop for 2D CNN inference [15]. The effect of quantizing 3D CNNs is yet to be carefully studied, but we assume that similar results for 2D CNNs would hold for 3D CNNs. To avoid overflow, the data width of *psum* is 32-bit which is larger than the summation of input and weight data width (16-bit). The input size of C3D is 3×16×112×112, which is the same as [5].

As we expected, both *input_chunk* and *psum_chunk* reach the largest size (196 KB and 784 KB) in the first layer of C3D. The *input_chunk* size here is affordable on most FPGA accelerators which typically include several megabytes (MB) on-chip block RAM. Note that *input_chunk* size is proportional to H, W, and D, which indicates that it will be enlarged on larger 3D CNNs. For instance, I3D [16] takes 64 frames as input, and the frame size is 224×224. Therefore, the *input_chunk* size in the first layer of I3D is 3,136KB. Nevertheless, Xilinx VC707 board has more than 4,500KB on-chip block RAM, which can fit that amount of data. Moreover, there are much larger (up to 40MB) on-chip memory (block RAM and UltraRAM) on Ultrascale+ devices. Thereby, grouping input data into $H \times W \times D$ size is a simple, efficient, and affordable strategy on modern devices.

Although the *psum_chunk* is four times larger than *input_chunk*, it is not necessary since it can be accumulated inside a *Multiply Accumulate* (MAC) unit when proper loop order is adopted. This will be discussed further in Section II-B. The *filter_chunk* is relatively much smaller, hence more *chunks* can be tiled in C and M dimensions.

B. Loop Ordering

We leverage the non-overlapping data tiling method in this paper, which groups input features and weights into *chunks*. Here we present the order of dimensions in which the *chunks* are fetched.

Fig. 4 depicts three distinct loop ordering strategies based on non-overlapping data tiling method. As mentioned above, there are only two dimensions left, C and M. If *Input Channel* (*IC*) C_t is fetched first, C_t channels of *psum_chunks* are calculated by convolving *filter_chunks* with *input_chunks* channel by channel (red arrow in Fig. 4). Strategy *IC* exploits *feature parallelism,* and C_t channels of *psum_chunks* can be accumulated to produce one channel of *outputs*. Hence only one *psum_chunk* needs to be buffered on the chip, as listed in Algorithm 1. On the contrary, when *Output Channel* (*OC*) M_t is fetched first, M_t numbers of *psum_chunks* are calculated by convolving M_t numbers of *filter_chunks* with the same *input_chunk* (green arrow in Fig. 4). Strategy *OC* takes advantage of *filter parallelism*, which reuses input features M_t times on different filters and reduces input data accesses consequently. However, strategy *OC* makes a request for buffering M_t numbers of *psum_chunks*.

Both strategy *IC* and *OC* cache *psum_chunk* on the chip which is four times larger than *input_chunk,* as mentioned in Section II-A. There may not be enough memory for caching *psum_chunk* on resource-constrained platforms. Under this circumstance, accelerator calculates each output pixel directly by performing $R \times S \times T \times C$ multiply-accumulates with C channels of *filter_chunks*, as listed in Algorithm 1. This strategy consumes *psums* in MAC units, and *No Partial sum* (*NP*) will be cached on the chip (blue arrow in Fig. 4). Strategy *NP* still needs to hold C channels of *filter_chunks* on the chip which typically occupies small capacity, as shown in Fig. 3.

Data reuse and memory access are affiliated to loop ordering. In the above strategies, filters are reused inside an *input_chunk*. By leveraging *feature parallelism* on channel dimension, *IC* reduces the memory footprint of *psum_chunks*. *OC* reuses input features on multiple *filter_chunks* through *filter parallelism*, thus reduces repeated accesses for input features. *NP* eliminates the demand for buffering *psum*, and also benefits from *filter parallelism* since the saved memory space can be used to cache more filters on M dimension.

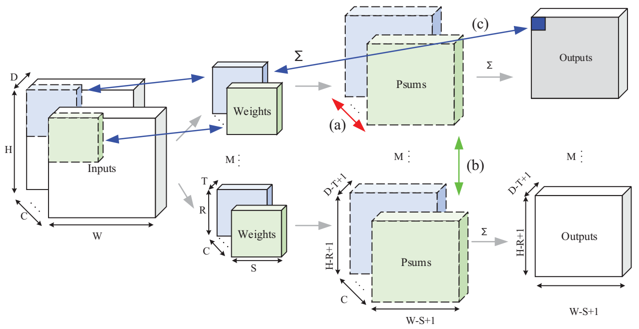

Fig. 4. Three distinct loop ordering strategies: (a) Input Channel first (IC); (b) Output Channel first (OC); (c) No Partial sum (NP).

Algorithm 1 *Loop Ordering Strategies*

```
calculate psum_chunk[m][c]:
  for (d, h, w) = (1, 1, 1) to (D−T+1, H−R+1, W−S+1) do
    psum = 0
    for (t, r, s) = (1, 1, 1) to (T, R, S) do
      psum+=input[c][d+t][h+r][w+s]*filter[m][c][t][r][s]
    psum_chunk[m][c][d][h][w] = psum
  return psum_chunk[m][c]

(a) Input Channel first (IC)
calculate output[m]:
  for m in 1 to M do
    for c in 1 to C do
      calculate psum_chunk[m][c]
      output[m] += psum_chunk[m][c]
    return output[m]

(b) Output Channel first (OC)
calculate output:
  for c in 1 to C do
    for m in 1 to M do
      calculate psum_chunk[c][m]
    output[1 to M] += psum_chunk[c] [1 to M]
  return output

(c) No Partial sum (NP)
calculate output[m][d][h][w]:
  for m in 1 to M do
    for (d, h, w) = (1, 1, 1) to (D−T+1, H−R+1, W−S+1) do
      output = 0
      for (c, t, r, s) = (1, 1, 1) to (C, T, R, S) do
        output+=input[c][d+t][h+r][w+s]*filter[m][c][t][r][s]
    return output[m][d][h][w]
```

Section II-A has demonstrated that the hyperparameters of 3D convolution vary across layers, so the benefits of different loop ordering strategies can fluctuate. It is optimal to choose the best performing loop order for each 3D CNN layer.

III. Hardware Architecture

In this section, we present the hardware architecture design. As shown in Fig.5, the accelerator consists of *Processing Element* (PE) arrays, reconfigurable on-chip SRAM, *Network-on-Chip* (NoC), controller, and off-chip DRAM.

A. Processing Element Array

The accelerator is built with N PE arrays, where each PE array is configured with R_a rows and C_a columns ($R_a = C_a = 3$ in Fig. 5). The PEs inside a PE array are cascaded in both row and column directions. Thus on-chip bus bandwidth can be reduced by passing input data to adjacent PEs. Each PE consists of a MAC, data buffers, and control logic. We adopt triple buffering in PEs to exploit both spatial and temporal data locality.

When performing 3D convolution, PE array shares the same weight among the PEs and transmits input features horizontally and vertically to behave like a sliding window. Consequently, $R_a \times C_a$ *psums* or *outputs* will be produced on a PE array. With the row buffer and column buffer inside a PE, a certain amount of spatial data can be reused when switching rows or columns. Data on the temporal dimension can be prepared and transferred into the temporal buffer when spatial data is reused. Thus the data transfer time can be overlapped with computing time. When the data transfer time can be perfectly hidden by the computation time, $R_a \times C_a$ *psums* or outputs will be calculated in $R \times S \times T$ cycles (*IC* and *OC*) or $R \times S \times T \times C$ cycles (*NP*). The output register of PE is cascaded in the column direction, and it requires R_a cycles to output $R_a \times C_a$ *psums* or outputs, which can also be overlapped with the computation time. This design strategy avoids broadcasting output which will significantly increase bus width and bandwidth, and lower the frequency.

To support different loop ordering strategies mentioned in Section II-B, different PE arrays can be assigned to calculate different channels (*IC*) or numbers (*OC*) of *psums*, or calculate the outputs directly with filters in size of $R \times S \times T \times C$ (*NP*). To maximize the computational efficiency, we should set the number of PEs inside a PE array to a factor of output feature size.

For *Fully Connected* (FC) layers, inputs are shared among the PEs, while weights are transferred individually since only inputs can be reused. Matrix multiplication is a memory-bound operation. Therefore, the bottleneck of processing FC layers is the off-chip memory bandwidth.

Arithmetic Logic Unit (ALU) in PE array accumulates *psums* from different channels when performing *IC* and *OC* strategies. Besides, ALU is also in charge of layers except for convolution and FC, such as relu, pooling, and downscaling (scale down output data width from 32-bit to 8-bit). There are C_a ALUs connected to the last row of PE array or *psum* data bus. Therefore, the input data rate matches with the output data rate of PE array.

B. Reconfigurable On-chip Memory

The optimal data tile size (including inputs, weights, and *psums*) varies across convolutional layers. To minimize on-chip memory fragmentation, we adopt the reconfigurable memory design strategy in [8] and modify it to FPGA platform (as shown in Fig. 6). The design is only used in the last-level SRAM. Each block is made up of several 36Kb block RAMs on FPGA, and blocks are allocated to each data type contiguously. Configuration registers are configured at each layer start time and denote the range of blocks used to store each data type. Configurable FSM is used to generate data access patterns into address generators. The traffic between on-chip memory and PE arrays are implemented through NoC. The NoC supports unicast

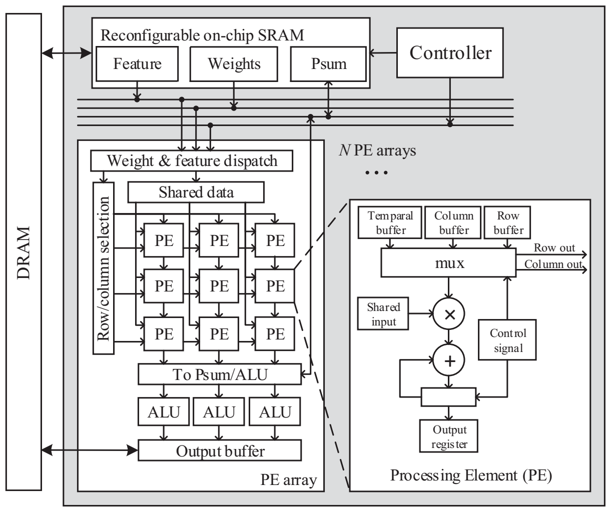

Fig. 5. Proposed accelerator architecture for 3D CNNs.

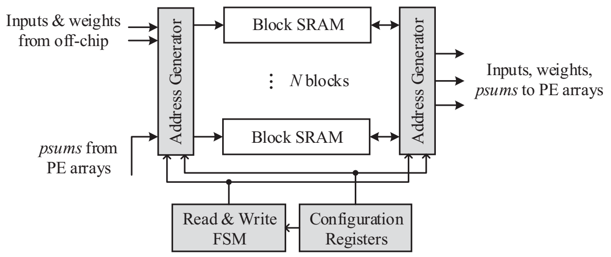

Fig. 6. Reconfigurable on-chip SRAM with N blocks.

and broadcast mode for different data parallelism introduced by different loop ordering strategies.

IV. Software Optimization

In this section, we describe a software optimization framework that finds the optimal tiling parameters and loop ordering strategies for each 3D CNN layer. The software optimizer takes the hyperparameters of 3D CNNs and the accelerator parameters as input. Then it enumerates all possible configurations to evaluate energy and performance results for each loop ordering strategy. Once all results are available, the optimal configuration for energy or performance is produced.

A. Configurations

To perform software optimization, the optimizer requests all the parameters related to energy consumption and performance:

1) 3D convolution hyperparameters: H, W, D, C, M, R, S, T, strides, padding, and data width.

2) Accelerator parameters: DRAM bandwidth, PE array size ($R_a \times C_a \times N$), on-chip SRAM capacity, data bus bandwidth, and frequency.

As mentioned in Section II, only C and M dimensions are left for fetching *chunks*. Consequently, there are six tiling parameters: *input channel* C_t (i_C_t), *filter channel* C_t (f_C_t), *filter number* M_t (f_M_t), *psum channel* C_t (p_C_t), *psum number* M_t (p_M_t), and *output channel* C_t (o_C_t). Nevertheless, when performing any of the three loop ordering strategies on the accelerator, the channels of *psum_chunks* are accumulated in

TABLE II. ENERGY COSTS OF OPERATIONS

8-bit Add	8-bit Mul	32-bit Add	32-bit Mul	32-bit SRAM Read (8KB)	32-bit DRAM Read
0.03pJ	0.2 pJ	0.1 pJ	3.1 pJ	5 pJ	640 pJ

ALUs, so that parameter p_C_t should not be considered. Another observation is that o_C_t is equal to p_M_t. Therefore, four tiling parameters (i_C_t, f_C_t, f_M_t, and p_M_t) should be considered when performing memory optimization.

B. Energy Consumption and Performance

The overall energy consumption of performing a convolutional layer can be calculated by (3):

$$E_{overall} = E_{DRAM\_access} + E_{L2\_access} + E_{L1\_access} + E_{compute} \quad (3)$$

where E_{DRAM_access}, E_{L2_access}, E_{L1_access}, and $E_{compute}$ are the energy consumption of DRAM access, reconfigurable on-chip memory (L2) access, local buffer (L1) access, and computation, respectively. Once the tiling parameters are assigned, there is enough information to compute the number of reads/writes that occur in L1, L2, and DRAM, the number of operations performed in each PE, and the ratio of PE utilization. We use an energy model to convert the number of reads/writes/operations to the expected energy consumption. For the energy model, energy numbers are taken from [17] and listed in Table II. Notice that DRAM energy per access is two orders of magnitude larger than SRAM, and three orders of magnitude larger than 8-bit MAC operation. Therefore, the energy consumption is dominated by memory access. Note that these optimizations need only be performed once per CNN. After optimal parameters are found, a configuration file can be saved and recalled instead of re-running the analysis.

V. EVALUATION

A. Experimental Setup

As a case study, we evaluate our data tiling and loop ordering strategies on three representative 2D and 3D CNN models: AlexNet [1], VGG16 [2], and C3D [5]. Since 2D convolution is a special case of 3D convolution with $D = T = 1$, our design strategy and software optimization can also be applied to 2D CNNs. We evaluate our hardware design on a Xilinx VC707 board with parameters listed in Table III.

B. Experimental Results

Fig. 7(a) compares the *optimal* (*opt*) model with baseline models (fixing strategy *IC*, *OC*, or *NP* across all layers) in terms of energy consumption in convolutional layers of C3D. The result demonstrates that fixing one configuration across all layers is sub-optimal, and it is necessary to flexibly support optimal configuration on each layer for better energy efficiency. Besides, the on-chip memory access energy dominates the overall energy in the optimal model. Fig. 7(b) gives more insight on how the L2 SRAM is partitioned between inputs, weights, and outputs for the optimal configuration in Fig. 7(a). Note that inputs consume more space in early layers, whereas filters occupy a larger space in later layers.

Table IV lists the normalized overall energy and DRAM accesses for all layers of C3D. Compared to baseline models, the

TABLE III. ACCELERATOR PARAMETERS

Platform	Xilinx VC707
DRAM	1GB DDR3 1,600 MT/s 64-bit
L2 (block SRAM)	1.125MB (36Kb × 8 × 32)
L1 (local buffer)	96KB
PE array size	7×7
PE array numbers	48
frequency	160 MHz
DSPs	2688 (96%)
BRAM	260.5 (25.3%)

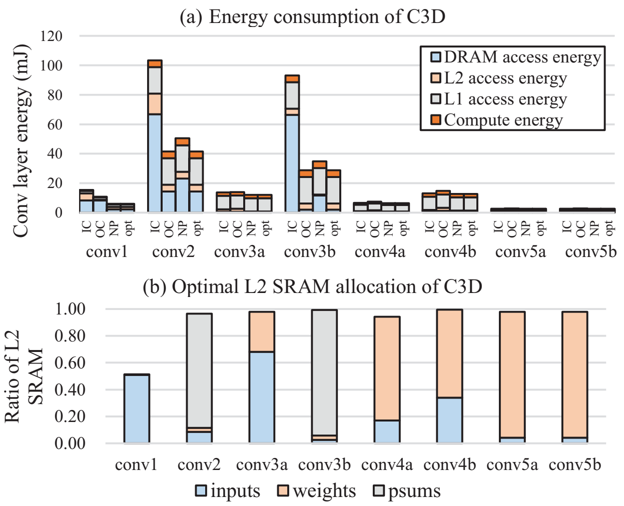

Fig. 7. (a) Energy consumption comparison in convolutional layers of C3D. (b) L2 SRAM allocation across layers of C3D for the optimal configuration.

TABLE IV. NORMALIZED OVERALL ENERGY AND DRAM ACCESSES

Loop strategy	*IC*	*OC*	*NP*	*opt*
Energy consumption	2.22x	1.09x	1.13x	1.00x
DRAM accesses	6.24x	1.26x	1.78x	1.00x

optimal model reduces 84%, 21%, and 44% DRAM accesses and 55%, 8%, and 12% energy consumption, respectively. Experiments for on-chip SRAM size and PE size are shown in Fig. 8. Strategy *OC* outperforms other strategies on resource-constrained hardware by leveraging *filter parallelism*. Since *filter parallelism* can also be exploited in multiple PE arrays and all *psums* are consumed inside PEs, strategy *NP* benefits more from PE resources and outperforms strategy *OC* on a larger system. *Feature parallelism* exploited by strategy *IC* shows less efficiency compare to *filter parallelism*. Due to the *input_chunks* cannot be reused and each PE array requests different *input_chunks* when performing *feature parallelism*, the off-chip memory accesses and the on-chip data bandwidth are increased significantly.

Fig. 9 presents the throughput performance of each layer with optimal configurations. The results are evaluated on Xilinx VC707 board with parameters listed in Table III. The peak throughput of our accelerator at 160 MHz is 860 GOP/s. Measured performance on convolutional layers reaches up to 753 GOP/s. The PE array is under-utilized in FC layers due to the DRAM bandwidth restriction. Hence the overall throughput is only 291 GOP/s on AlexNet, while the average throughput of the convolutional layers is 676 GOP/s.

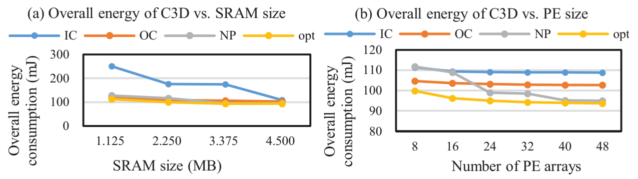

Fig. 8. (a) Correlation between overall energy of C3D and SRAM size (48 PE arrays). (b) Correlation between overall energy of C3D and PE size (4.5MB SRAM).

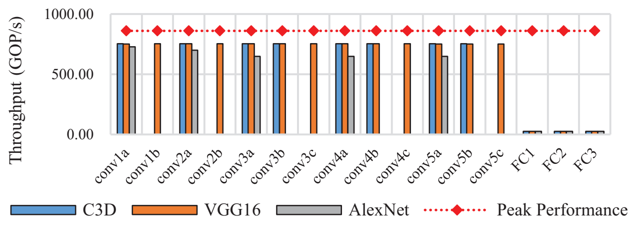

Fig. 9. Throughput performance of each layer in C3D, VGG16, and AlexNet.

Table V compares our design with state-of-the-art works. As different FPGA platforms and DSP slices are used, we list the performance density (the number of arithmetic operations that one DSP executes in one cycle) for fair comparisons. Note that a MAC is counted as two operations. Results demonstrate that our design achieves 63% more performance density than [10] which also adopts data tiling method. Mapping convolutions to matrix multiplication in [14] shows 3% more performance density in VGG, whereas 7% behind in C3D since a higher degree of data replication is introduced in 3D convolution. The Winograd algorithm implemented in [13] reduces the multiplications in 3D convolution by 70.4% at the cost of 101% more additions. The additions are executed with LUTs instead of DSP slices. That is why the implementation in [13] achieves the best performance density. However, their design can only be applied to fixed kernel size and convolutional stride, so that reprogramming FPGA is needed to support different CNN models. By comparison, our design is more generic and can be adopted to various models with state-of-the-art performance.

VI. Conclusion

This paper presents a design space exploration of memory access optimization for accelerating 3D CNNs on FPGA. With the non-overlapping data tiling method and adopting the optimal loop order for each 3D CNN layer, the memory overhead and the off-chip memory accesses can be reduced significantly. Consequently, the overall energy efficiency and the computing unit utilization are increased, which is crucial for designing a highly efficient 3D CNN accelerator.

Acknowledgment

This research was sponsored by Huawei Innovation Research Program (HO2018115161). Supports from Qiannan Lv, Zhenhua Yang, and Gang Li are gratefully acknowledged.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

TABLE V. Comparison with Other FPGA Implementations

	[7]	[10]	[13]	[14]		Ours	
FPGA	Stratix V GXA7	Xilinx ZC706	Xilinx VC709	Xilinx VC709		Xilinx VC707	
CNN Model	VGG	C3D	C3D	VGG	C3D	VGG	C3D
Precision	16-bit fixed	16-bit fixed	16-bit fixed	8-16 bits fixed		8-bit fixed	
Clock (MHz)	150	172	150	120		160	
DSPs	1568 (100%)	810 (90%)	1536 (42%)	3595 (99%)		2688 (96%)	
Latency (ms)	88.8	542.5	89.4	44.8	115.5	46.9	107.9
Overall throughput (GOP/s)	349	142	431	692	668	660	714
Performance Density (OP/DSP/cycle)	1.48	1.02	1.87	1.60	1.55	1.53	1.66

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[3] Y. Taigman, M. Yang, M. A. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701-1708.

[4] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence,* vol. 35, no. 1, pp. 221-231, 2012.

[5] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489-4497.

[6] Q. Dou *et al.*, "Automatic detection of cerebral microbleeds from MR images via 3D convolutional neural networks," *IEEE transactions on medical imaging,* vol. 35, no. 5, pp. 1182-1195, 2016.

[7] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, no. 7, pp. 1354-1367, 2018.

[8] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible Acceleration for 3D CNN-based Video Understanding," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018: IEEE, pp. 933-946.

[9] K. Hegde *et al.*, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018: IEEE Press, pp. 674-687.

[10] H. Fan, X. Niu, Q. Liu, and W. Luk, "F-C3D: FPGA-based 3-dimensional convolutional neural network," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017: IEEE, pp. 1-4.

[11] T. Geng *et al.*, "FPDeep: Acceleration and load balancing of CNN training on FPGA clusters," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018: IEEE, pp. 81-84.

[12] T. Geng, T. Wang, A. Li, X. Jin, and M. Herbordt, "A Scalable Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Weight and Workload Balancing," *arXiv preprint arXiv:1901.01007,* 2019.

[13] J. Shen *et al.*, "Towards a uniform template-based architecture for accelerating 2d and 3d cnns on FPGA," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018: ACM, pp. 97-106.

[14] Z. Liu *et al.*, "A uniform architecture design for accelerating 2D and 3D CNNS on FPGAs," *Electronics,* vol. 8, no. 1, p. 65, 2019.

[15] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342,* 2018.

[16] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6299-6308.

[17] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014: IEEE, pp. 10-14.