Energy-Efficient Dataflow Scheduling of CNN Applications for Vector-SIMD DSP

WONTAE KIM1, SANGHEON LEE1, ILWI YUN1, CHULHEE LEE2, KYUJOONG LEE3, (Member, IEEE), AND HYUK-JAE LEE1, (Member, IEEE)

1Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
2System LSI Division, Samsung Electronics Corporation, Hwaseong 18448, South Korea
3School of AI Convergence, Sungshin Women’s University, Seoul 02844, South Korea

Corresponding author: Kyujoong Lee (kyujoonglee@sungshin.ac.kr)

This work was supported by Samsung Electronics Company Ltd.

ABSTRACT Dataflow-scheduling techniques for convolutional neural networks (CNNs) are extensively studied to minimize the off-chip memory access. However, the efficiencies of the previously proposed techniques are limited because their optimizations only consider the general hardware such as FPGA and GPU. To overcome this limitation, this paper proposes dataflow scheduling for vector-SIMD DSP to minimize the energy consumption for the off-chip memory access. First, the proposed technique attempts to group as many given layers as possible. For grouping the layers, the tiles in different layers are executed in sequence without the off-chip memory access except the first and the last layers in the group. The length of the grouped layers is determined with regard to the minimization of the energy consumption of off-chip memory by estimating the proposed energy model of the off-chip memory. However, grouping the layers results in the additional computation. To minimize this overhead, this paper solves the optimization problem for in the grouped layers. Second, for layers that cannot be grouped, the tiling along the W-axis is not considered, to maximize the size of the overlapped data in consecutive tiles. Consequently, the reuse of the overlapped data in the on-chip buffer is maximized, thereby reducing the energy consumption by the off-chip memory. For evaluation, a cycle-accurate simulation environment is established to measure the energy consumption of the off-chip memory by tracing the data between a vector-SIMD DSP and an off-chip memory. The experimental results show that compared with the baseline tiling and scheduling techniques, the proposed technique reduces the energy consumption by an average of 51% for CNN applications such as Tiny YOLOv2, MobileNetv1, VDSR.

INDEX TERMS DSP, SIMD, CNN, memory access, energy consumption reduction.

I. INTRODUCTION Convolutional neural networks (CNNs) have successfully solved numerous computer vision tasks such as image detection [1], [2], recognition [3], [4], and segmentation [5], [6]. Extensive efforts have been made toward the development of neural processing units (NPUs), which offer high performance and energy efficiency for convolution operations. NPUs are widely used in not only high-performance server systems but also low-power embedded systems. For battery-powered embedded systems, energy efficiency is critical to extending the operation period of the system. In addition, it is reported that more than 80% of the total energy is consumed via off-chip memory accesses [7], [8]. Therefore, the minimization of off-chip memory accesses is important to reduce the energy consumption of embedded systems with NPUs.

Many NPUs have been developed using FPGAs [9], [10], [11], [12], [13], GPUs [14], DSPs [19], [20], and ASICs [7], [15], [16]. Recently, DSP vendors such as Cadence and CEVA presented DSPs with an NPU dedicated for CNN inference. These DSPs accelerate the convolution operations by using the SIMD (single instruction multiple data) architecture to achieve high performance and energy.
efficiency in CNN inference. The automated tool for DSPs converts a floating-point-based program into an optimized, fixed-point-based DSP code. It provides several optimization techniques to reduce the computation time, i.e., local memory management, DMA (direct memory access) management, quantization process. However, these optimization techniques cannot minimize the data transfer between a DSP and an off-chip memory.

Dataflow scheduling has been widely used to reduce the off-chip memory access in an embedded NPU system because the size and type of the data to be stored in an on-chip buffer of an NPU are determined according to the data scheduling [11], [12], [13], [17], [18], [21]. In a CNN, there are the following three types of data: input features, output features, and weights. The study regarding dataflow scheduling is classified into two cases. The first case is to modify the nested loop for the convolution operation. In this case, the optimal data type is selected to minimize the off-chip memory access for each layer according to the characteristic of the memory system. In [12], the optimal tiling factor for a computational-to-communication ratio was selected among all the possible tiling factors in the nested loop. However, the selected tiling factor is a sub-optimal because the reuse of only the output feature map is considered in the communication term. However, in [13], all the reuse possibilities of the input feature map, output feature maps, and weight were examined, and a tiling factor with the minimal off-chip memory access was calculated for each layer. Furthermore, in [18] the size of the on-chip buffer was considered to estimate the optimal tiling factor. However, these studies did not consider reusing the redundant data between layers for the layer-wise dataflow scheduling. The second case is to reuse the overlapped data in the tiled convolution operation [17]. In [11] and [21], a method was proposed to reuse the overlapped data in adjacent tiles and in adjacent layers, respectively. In addition, in [11] a tiling factor, which maximized the amount of the redundant data that occurred between successive tiles, was estimated, depending on the access pattern of the input feature map; subsequently, the data were reused in the on-chip buffer. In [21], a method was proposed to reuse the overlapped feature map data between consecutive layers in an on-chip buffer. Generally, in the case of layer fusion, the instances of off-chip memory accesses can be further reduced compared with those in the first methods; however, this process increases the execution time of the convolutional layer because of redundant convolution operations.

In addition, [23] and [30] generates automatically the optimal dataflow scheduling by considering the target hardware. [23] presented an analytical model called MAESTRO (Modeling Accelerator Efficiency via Spatio-Temporal Reuse Occupancy) to estimate the execution time, the energy efficiency, and the hardware cost of dataflows. Specifically, MAESTRO finds the optimal method by simulating with various combinations of data reuse methods and mapping methods on the computing unit. As the results, [23] offers the opportunity to achieve both higher performance and energy efficiency to the accelerator designers. Reference [30] only focuses on the implementation of CNN applications in DSP architecture. However, the automation tool [30] of Cadence does not consider the characteristics of layers, i.e., the data scheduling and the tile size are always fixed. Therefore, for some layers, the instances of off-chip memory accesses are not reduced.

This study presents a dataflow scheduling method to minimize the energy consumption of the off-chip memory in an embedded NPU system, especially a vector-SIMD DSP system. The following are the main contributions of this study:

1) Depending on the characteristics of given layers, the proposed method determines how the layer-fusion technique can be employed in the vector-SIMD DSP systems using the proposed energy-consumption model of the off-chip memory.
2) For both the original layer-wise dataflow and grouped dataflow via layer fusion, the optimal tile sizes are obtained using the mathematical model.
3) A cycle-accurate simulation environment is implemented to measure the off-chip memory access by tracing the data between an DSP and off-chip memory. CNN applications such as Tiny YOLOv2, MobileNetv1, and VDSR are simulated in this environment. The simulation results show that the proposed technique reduces the energy consumption by an average of 51%.

The rest of this study is organized as follows. Section II describes the previously conducted studies, a background about embedded NPUs, and the motivation behind this study. Section III proposes the dataflow scheduling and tiling technique to minimize the instances of off-chip memory accesses. Section IV presents the implementation of the overall simulation environment, and it also evaluates the proposed method and the baseline. Section V concludes this study.

II. BACKGROUND AND MOTIVATION

A. CONVOLUTION ON VECTOR-SIMD DSP

A vector-SIMD DSP comprises hundreds of MACs. For using these MACs efficiently, DSP vendors provide the optimized convolution-operation library to use the MAC array because the high utilization of the MAC array increases the processing...
speed of a CNN application. For example, Cadence offers two optimized convolution libraries for the Tensilica Vision P6 DSP [19], which comprises 256 MACs. Fig. 1 depicts two types of optimized convolution that utilize the entire MAC array at once using 64-way SIMD operations through a computational graph. The size of the MAC array is 64 × 4 (256) in Vision P6 DSP. These two optimized 4-Tap convolution and Quad-MAC convolution perform four multiplication and one accumulation in a clock cycle. In Fig. 1, vec and wvec represent a 64-way 512-bit vector register and a 64-way 1,536-bit wide-vector register for accumulation, respectively. As depicted in Fig. 1 (a), two 512-bit feature-map data are stored in vec0 and vec1. These two registers are concatenated and, subsequently, shifted by 0, 1, 2, and 3 bytes. Each Tap multiplies the shifted vector register by 1 byte of different weight data, following which the results of each Tap are accumulated to wvec. Consequently, this 64-way SIMD operation achieves the same result as 1D convolution. However, there is a case where the last 4th Tap (Tap 3) operation is meaningless operation. Because the kernel size of the weight filter is mostly 3 × 3, only three out of four weight data are multiplied by feature data for 1D convolution. Therefore, valid operations are processed for only 192 of the 256 MACs. In Fig. 1 (b), four 512-bit weight data are stored in vec0, vec1, vec2, and vec3 registers. Each Tap multiplies each vector register by 1 byte of the feature map data, following which the results of each Tap are added to wvec. Unlike in Fig. 1 (a), all the MAC operations in Fig. 1 (b) are valid. Because 64 different weight data are required for one feature data, the channel depth of the output feature map should be greater than or equal to 64 to use the convolution operation depicted in Fig. 1 (b). Fig. 2 depicts the tiling of both input feature map and output feature map when a DSP performs a convolutional operation in Fig. 1 (b). Fig. 2 depicts the tiling of both input feature map and output feature map when a DSP performs a convolutional operation. However, convolution results of tile1 in the last layer, the computation for the overlapped region is performed repeatedly, except in the last layer. The tile size of the grouped layers is determined on the basis of the tile size of the last layer. For example, if the size of output tile0 in the third layer is \((tw_3, th_3)\), the size of input tile0 is \((tw_3 \cdot s_3 + f_3 - 1, th_3 \cdot s_3 + f_3 - 1)\) when the filter size and the stride factor are \(f_3\) and \(s_3\), respectively. Consequently, input tile0 in the third layer becomes output tile0 in the second layer, and the size of input tile0 in the second layer becomes...
\( (s_2 \cdot tw_2, s_2 \cdot th_2) \) when the pooling stride is \( s_2 \). Similarly, input tile0 in the second layer becomes output tile0 in the first layer, and the size of input tile0 in the first layer is \((tw_1 \cdot s_1 + f_1 - 1, th_1 \cdot s_1 + f_1 - 1)\). In the process of tile1, the operation previously performed for tile0 should be performed again. In Fig. 3, the feature map data of overlapped region at the boundary of tiles0 and tile1 is used multiple times for the convolution operation. And the size of overlapped region gradually increases as it reaches the first layer in the grouped layers. is largest in the first layer among the grouped layers. In other words, the longer the group length, the larger the size of overlapped region in the grouped layers. Thus, the proposed method should consider the relationship between the reduced off-chip memory access and increased processing time for the overlapped computation. In the case of ResNet [35] style, the input tile data of every residual block should be additionally stored in the on-chip buffer for the residual operation. Therefore, if DSP has the same on-chip buffer, the number of layers that can be fused in the ResNet case should be less than in the normal CNN without skip connections.

For the overlapped computation, [21] uses a reuse scheme that copies the overlapped data to next tile’s on-chip buffer, which causes in no additional processing time. This is because the latency for the data transfer can be hide during the current tile’s computation. On the other hand, a recomputing scheme that newly computes the overlapped data every time is suitable for vector-SIMD DSP. This is because the efficiency of the vectorized convolution in the DSP is reduced when the reuse scheme is applied to the DSP. For the DSP, the tile size is closely related to the efficiency of convolution operation. In order to use the reuse scheme, the original optimal tile size should be changed because the overlapped data to be used next must be stored in the on-chip buffer. As a result, the processing time is longer than before applying the reuse scheme.

There are several studies using the layer fusion in the various environment [22], [24], [25]. Reference [22] introduces a tool that performs various optimizations by analyzing the computational graph of the existing deep learning framework by considering when the target platform is CPU, GPU, and FPGA. In [22], the operator fusion is used as the layer fusion to reduce the off-chip memory access. Especially, [22] reports, in GPU platform, there is no additional execution time that occurs due to the overlapped computation, but rather, due to the operation fusion, several operation kernels are replaced with single kernel to reduce execution time.

Layer fusion is also widely used for a distributed system because it can reduce the data transfer between edge devices. References [24] and [25] performed parallel execution of CNN inference by using the layer fusion. In the normal layer-wise dataflow, a hub should distribute the input data to the edge devices, and receive the output data back from the edge devices per layer. However, in the layer-group-wise dataflow, the hub does not need to collect the output data for each layer. Therefore, the layer fusion method can reduce the communication cost between the hub and the edge device.

\[ \text{FIGURE 4. Example of data movement in a vector-SIMD DSP system according to dataflow. (a) Layer-wise dataflow and (b) layer-group-wise dataflow.} \]

C. OPPORTUNITY TO REUSE REDUNDANT DATA

Fig. 4 (a) depicts an example of the data movement between registers, on-chip buffers, and an off-chip memory in a DSP memory system. \( i \)-tile and \( o \)-tile denote the input tile and output tile, respectively, and \( i \)-tile\(_0\) denotes the zeroth \( i \)-tile in the \( n \)-th layer. And it is assumed that \( i \)-tile\(_0\) and \( i \)-tile\(_1\) are adjacent to each other. The dataflow of the tile in a layer is described as follows. First, a DSP requests \( i \)-tile\(_0\) from the off-chip memory to store \( i \)-tile\(_0\) into the on-chip buffer by DMA (3); subsequently, the DSP transfers a part of the \( i \)-tile\(_0\) stored in the on-chip buffer to the register file (2), following which it performs the required operations such as convolution and max pooling (3). Upon completing the operation for the part of the \( i \)-tile\(_0\) stored in the register file, the corresponding results are returned to the on-chip buffer (4). After the operation for all \( i \)-tile\(_n\) is completed, all the \( o \)-tile\(_n\)bs stored in the on-chip buffer are transferred to the off-chip buffer by DMA (5). The operations (1 – 5) are repeated until the operation of all \( i \)-tile is completed. In this repeated process, there are two possibilities of data reuse. The first possibility occurs when storing \( o \)-tile in the off-chip memory (5). The \( o \)-tile\(_0\) and \( o \)-tile\(_1\) stored in the on-chip buffer of the \( n \)-th layer are, respectively, the same as the \( i \)-tile\(_{n+1}\)0 and \( i \)-tile\(_{n+1}\)1 stored in the on-chip buffer of the \((n + 1)\)-th layer (yellow region and green region in Fig. 4 (a)). Therefore, the instances of the accesses to the off-chip memory can be reduced if the operation for the \( o \)-tile\(_0\) in the \((n + 1)\)-th layer is directly performed without storing \( o \)-tile\(_0\) to the off-chip memory in \( n \)-th layer. In addition, the second possibility of data reuse occurs when the DSP performs the convolution for the consecutive tiles \( i \)-tile\(_0\) and \( i \)-tile\(_1\). At this time, there is the data called repeatedly by the kernel operation in the consecutive tiles (red region in Fig. 4 (a)). The size of data called repeatedly is basically dependent on the size of the tile.

This study proposes two reuse schemes to utilize these possibilities of data reuse. The first reuse scheme groups the successive layers to not access the off-chip memory in
TABLE 1. Comparison between [21] and the proposed method.

| GL                        | GB transferred | $E_{\text{sim}}$ (mJ) |
|----------------------------|----------------|------------------------|
| Baseline 1,1,1,1,1,1      | 32.845         | 4.490                  |
| [21] 4,1,1,1,1,1          | 19.450         | 3.067                  |
| Prop. 3,1,1,1,1,1         | 20.779         | 2.780                  |

the grouped layers, which denotes **inter-layer data reuse**. The second reuse scheme reuses the overlapped feature data between tiles, which denotes **inter-tile data reuse**. If **inter-layer data reuse** is applied from $n$-th layer to $m$-th layer, the off-chip memory is not accessed between the load of $i$-tile of the $n$-th layer to the store of $o$-tile of the $m$-th layer. The weights of the grouped layers are assumed to be stored in the on-chip buffer before performing the convolution operation. However, overlapped computation occurs in the grouped layers, increasing the processing time compared with the processing time via computation without layer fusion. Thus, the proposed method determines the optimal tile size by using a mathematical model for the minimization of the overlapped computation. Furthermore, in **inter-tile data reuse**, the optimal tile size for a single layer without **inter-layer data reuse** is determined to reuse the overlapped data to a feasible extent. Therefore, the proposed method models the size of the overlapped data according to the tile size, following which it selects the tile size to maximize the size of the overlapped data between successive tiles. Table 1 shows the short comparison results when the reuse schemes are used. **GL** denotes the set of the group length. $E_{\text{sim}}$ denotes the energy consumption of the off-chip memory measured by the proposed simulation environment. Reference [21] uses the layer fusion method as **inter-layer data reuse** in terms of the off-chip memory data transfer, however, the proposed method does in terms of the energy consumption of the off-chip memory. Compared to baseline, both methods using the layer fusion shows superior results in both data transfer and the energy consumption. However, depending on the layer fusion method such as fusing first four convolutional layers or first three convolutional layers, the results show the difference in terms of the data transfer and the energy consumption. This paper searches an optimal **GL** in terms of the energy consumption in order to minimize the additional computation by **inter-layer data reuse**.

The energy consumption of the off-chip memory can be reduced by two reuse schemes: **inter-layer data reuse** and **inter-tile data reuse**. However, if two reuse schemes are feasible, the proposed method determines which of the two schemes can further reduce the energy consumption of the off-chip memory. Generally, **inter-layer data reuse** is advantageous because the entire tile data is reused in the on-chip buffer. Therefore, the proposed method attempts to use **inter-layer data reuse** to a feasible extent. The goal of the algorithm in Subsection III. A is to maximize the number of the layers that are tiled using **inter-layer data reuse** while minimizing the processing time.

![FIGURE 5. Comparison of dataflows. (a) Layer-wise dataflow and (b) layer-group-wise dataflow.](image-url)

III. PROPOSED DATAFLOW SCHEDULING FOR DSP

A. LAYER-GROUP-WISE DATAFLOW

To optimize the off-chip memory access, the proposed method should employ the layer fusion to a feasible extent by analyzing the given CNN. If **inter-layer data reuse** is employed to the original layer-wise dataflow in Fig. 5 (a), it is converted to the layer-group-wise dataflow depicted in Fig. 5 (b). $L$ and $G$ denote the single layer and grouped layers, respectively. $n$ and $m$ denote the original number of the layers and the number of layers in the layer-group-wise dataflow, respectively. In $G$, the off-chip memory access is not necessary except for the **ifmp** of the first layer and the **ofmp** of the last layer. However, upon employing the layer fusion, the required computation increases in the remaining convolutional layers except for the last convolutional layer. Thus, the optimal size of $G$ should be determined by considering this trade-off. This paper proposes to determine the optimal size of $G$ in terms of the energy consumption of the off-chip memory.

Fig. 6 depicts the overall flow of grouping a given CNN. Several configuration parameters, such as feature-map size, timing parameter of the memory model, and computing resource, are required for grouping the given CNN. Considering the prior information, the proposed grouping algorithm finds the optimal parameters for the grouped length and the tile size offline before DSP runs CNN. Therefore, the computational complexity of the proposed grouping algorithm does not affect CNN processing time. After completing the grouping process, the layer-group-wise information and optimal tile size for each layer are generated. In Fig. 6, **GL** denotes the set of the group length. Each layer-group-wise dataflow has a different **GL**. $E_i$ denotes the estimated energy consumption of the off-chip memory. $E_{i,j}$ denotes $E$ from $i$-th layer to $j$-th layer. N and M denote the number of all possible group candidates, and the size of **GL**, respectively. The possible layer-group-wise dataflows are determined under the
condition that the size of all weight parameters in G does not exceed the size of the on-chip buffer. Tile denotes the set of tile size. OpGL and OpTile denote the optimal GL and Tile among all layer-group-wise dataflows in terms of the energy consumption of the off-chip memory.

The proposed algorithm stores the energy consumption of the off-chip memory for all layer-group-wise dataflows and later finds a layer-group-wise dataflow with the minimum energy consumption. Specifically, in order to get $E_{i,j}$ for $G_s$, the tile size in each $G_s$ should be determined first. For example, if the $m$-th group length is greater than 1, the tile size is determined by solving (5), and if not, the tile size is fixed to $(W, th, td)$. ((5) is explained on the next page) The detail of the tile size decision is explained to next section. And $E_{i,j}$ is calculated by using the proposed energy consumption model of the off-chip memory.

$$E_{i,j} = \min(E(1), \ldots, E(N))$$

$$OpGL = GL_i$$

$$OpTile = Tile_i$$

In (1), $E_{i,j}$ is derived using the mathematical model based on the power-consumption pattern when the DSP runs CNN applications. Generally, because the DSP system has limited computing resources, CNN applications are mostly compute-bound. Fig. 7 depicts the power-consumption pattern when CNN applications are compute-bound in the DSP system. $P_{act}$ and $P_{idle}$ denote the power during the read/write operation of the off-chip memory and the during the idle state of the off-chip memory, respectively. In addition, $T_{act}$ and $T_{idle}$ denote the time taken during the read/write operation of the off-chip memory and the time taken during the idle state of the off-chip memory, respectively. Notably, $T_{idle}$ is greater than $T_{act}$, and $T_{act} + T_{idle}$ is repeated periodically because tiled convolution should be performed because of the limited size of the on-chip buffer of the DSP. The blue bar and green bar denote the read and write operations of the off-chip memory, respectively. One has the following:

$$E_{i,j} = (P_{act} \cdot T_{act} + P_{idle} \cdot T_{idle}) \cdot N_{Tile} \tag{1}$$

where $E_{i,j}$ is calculated using $P_{act}$, $P_{idle}$, $T_{act}$, and $T_{idle}$. $N_{Tile}$ denotes the number of tiles in the grouped layers. $P_{act}$, $P_{idle}$, $T_{act}$, and $T_{idle}$ were set with [26]. And $T_{act}$ is calculated by the following:

$$T_{act} = (Fmp_{i,j} + Wght_{i,j}) \cdot \frac{Mem\_Cycle}{Mem\_Freq} \tag{2}$$

where $T_{act}$ is first calculated by multiplying the size of the requested data in the grouped layers by the processing time for the read/write operation. The size of the required data is the sum of $F_{i,j}$ and $W_{i,j}$. $Fmp_{i,j}$ and $Wght_{i,j}$ denote the size...
of feature map and weight data from layer $i$ to layer $j$, respectively. $Mem\_Cycle$ and $Mem\_Freq$ denote the read/write cycle and denotes the operating frequency of the off-chip memory, respectively. Also, $Mem\_Cycle$, $Mem\_Freq$ were set with [26]. Finally, $T\_idle$ is determined by the convolution operation time for a tile, and it will be discussed in the next section.

### B. LAYER-GROUP-WISE DATAFLOW

DSP supports hundreds of MAC operations per 1 clock cycle by using the single coupled MAC array, and the MAC operations are performed by the SIMD manner. For example, the efficiency of 64-way SIMD operation is dependent on loading the as much valid data as possible into a 64B vector register from the on-chip buffer. In other words, for the single layer, the tile width should be simply determined to a multiple of 64. However, for the grouped layers, the tile size decision process should consider the SIMD operation efficiency of all layers in the group at a time because all tile sizes in the grouped layers are coupled. Especially, because the grouping layers increases the number of SIMD operations as discussed in Section 3, the tile size decision to minimize this additional computation is essential.

For the grouped layers, this paper defines the number of SIMD operations according to the possible tile size, as shown in (3) and (4). Notably, (3) is for $WHD$, whereas (4) is for $DWH$.

$$\begin{align*}
N\_SIMD(tw_i, th_i, td_i) &= \left[ \frac{W_i}{tw_i} \right] \cdot \left[ \frac{tw_i}{64} \right] \cdot \left[ \frac{H_i}{th_i} \right] \cdot \left[ \frac{th_i}{2} \right] \\
&\quad \cdot \left[ \frac{D_i}{td_i} \right] \cdot \left[ \frac{td_i}{2} \right] \cdot 12 \cdot M_i \\
&= \left[ \frac{W_i}{tw_i} \right] \cdot \left[ \frac{tw_i}{64} \right] \cdot \left[ \frac{H_i}{th_i} \right] \cdot \left[ \frac{th_i}{2} \right] \\
&\quad \cdot \left[ \frac{D_i}{td_i} \right] \cdot \left[ \frac{td_i}{64} \right] \cdot 9 \cdot M_i
\end{align*}$$

(3)

(4)

where $N\_SIMD$ denotes the number of SIMD operations in the $i$-th layer. In addition, the terms $W_i$, $H_i$, $M_i$, and $D_i$ denote the width, height, input-channel depth, and output-channel depth of the $i$-th layer, respectively, and $tw_i$, $th_i$, and $td_i$ denote the tile width, height, and output-channel depth of the $i$-th layer, respectively. In the case of $WHD$, the output size is 64 × 2 × 2 ($W \cdot H \cdot D$), and in the case of $DWH$, the output size is 2 × 2 × 64 ($W \cdot H \cdot D$), for each SIMD-operation loop. $N\_SIMD$ is 25% less in $DWH$ than that in $WHD$, as mentioned in Subsection II. A. Fig. 8 shows an example of the nested loop for the tiled convolution in $WHD$. Notably, 4_Tap_convolution is a library that utilizes hundreds of MACs in one clock cycle in the SIMD manner; therefore, one 4_Tap_convolution is regarded as one SIMD operation. In $WHD$, 12 SIMD operations are performed in one input channel loop ($m$). Therefore, there is a constant 12 in (3).

To obtain the tile size that minimizes $N\_SIMD$, this paper formulates the following optimization problem for various networks:

$$\begin{align*}
\text{Problem) } & \quad \text{find } OpTile \\
& \quad \text{minimize } \text{Grouped}_\text{N}\_SIMD(tw_j, th_j, td_j) \\
& \quad \text{subject to: } \\
& \quad \quad Wght_{i-j} + Fmp_{i-j} \\
& \quad \quad 0 < tw_i < W_j, 0 < th_i < H_j, 0 < td_j < D_j
\end{align*}$$

(5)

(6)

(7)

The solution to this problem is $OpTile$, which denotes the optimal tile-size set of the grouped layers. Upon selecting the tile size of the last layer $(tw_j, th_j, td_j)$ in the grouped layers, the corresponding tile size of the remaining layers is determined as explained in Subsection II. B. The objective function is $\text{Grouped}_\text{N}\_SIMD$, which is the summation of $N\_SIMD$ in the grouped layers. One has the following:

$$\text{Grouped}_\text{N}\_SIMD(tw_j, th_j, td_j) = \sum_{n=1}^{j} N\_SIMD(tw_n, th_n, td_n)$$

(8)

where $\text{Grouped}_\text{N}\_SIMD$ denotes the overall number of SIMD operations in the grouped layers according to the possible tile size. However, (8) can be represented by the computation time, as well as the number of SIMD operations from the $i$-th layer to the $j$-th layer. Therefore, $T\_idle$ in Subsection III. A can be calculated using (7) as follows:

$$T\_idle = \text{Grouped}_\text{N}\_SIMD(tw_j, th_j, td_j) / \text{DSP}_\text{Freq} - T\_act$$

(9)
where $DSP\_Freq$ denotes the operating frequency of the DSP system. $Grouped\_N\_SIMD$ denotes not only the overall number of SIMD operations but also the number of total cycles required for that particular number of SIMD operations. This is because the time required for the 4_Tap_convolution operation is 1 clock cycle in the used DSP system. This processing time for the SIMD operation is estimated according to the computation resource of the DSP to be used; because the memory and SIMD operations are executed simultaneously, $T_{act}$ should be subtracted from $Grouped\_N\_SIMD$ to obtain only the idle time of the off-chip memory.

In the case of a single layer, the tile size is selected to minimize the off-chip memory access because of the occurrence of overlapped data. Moreover, the part of the $i$-tile currently stored in the on-chip buffer is required for the next $i$-tile in the same convolutional layer, as depicted in Fig. 9. The amount of the overlapped data in the boundary is not significant compared with that of the overall off-chip memory bandwidth. To reduce the amount of the overlapped data, $tw$ and $th$ should be increased to a feasible extent, when considering the size of the on-chip buffer. In addition, the tile should be of square shape, i.e., $tw=th$. Furthermore, the overlapped data between the two adjacent tiles can be reused in the on-chip buffer. (10) expresses the size of ifmp that the DSP requests to the off-chip memory when reusing the aforementioned overlapped data. Furthermore, one has the following.

\[
V\,(tw) = ([W/\,tw]\cdot[H/th]\cdot[M/tm]) \\
\cdot((tw+f-1)/th\cdot tm) \propto 1 + 1/tw 
\tag{10}
\]

where $V$ denotes the size of ifmp that the DSP requests to the off-chip memory when the size of ifmp is $(tw, th, tm)$ and the filter size is $f$. The first term denotes the number of tiles in ifmp and the second term the size of a single tile excluding the overlapped region $(tw+f-1)/th$. Except for the constant term $(W, H, M)$ and the ceiling in (10), only $(1/tw)$ remains as the variable. Therefore, $tw$ should be set to $W$ to minimize $V$, thereby maximizing the overlapped region of successive tiles. Consequently, the reuse of the overlapped region is also maximized. However, if the tile is processed in the horizontal direction, $th$ should be set to be the height of ifmp. Because $tw$ should be set to be smaller than 64, the efficiency of the SIMD operation is unavoidably reduced in the WHD structure. Therefore, $tw$ should be set to $W$.

Fig. 10 depicts an example of the optimal tile size in the single layer according to the structure of ifmp. Fig. 10 (a) and (b) depicts the ifmp in WHM and MWH storage patterns, respectively. The ifmp in WHM represents the array stored in the following order: $W$-direction, $H$-direction, and $M$-direction. However, the ifmp in MWH is stored in the following order: $M$-direction, $W$-direction, and $H$-direction. The yellow box represents the first $i$-tile and the blue one the second $i$-tile. Considering the reuse of the overlapped data, the optimal tile sizes for $tw$, $th$, and $tm$ are $W$, 4, and $M$, respectively. The tile size is the same irrespective of the ifmp structure. The size of the overlapped data is $W \cdot 2 \cdot M$ when the size of the weight filter is 3, and the overlapped data are described in black in Fig. 10. If not reusing the overlapped data, the $W \cdot 4 \cdot M$ i-tile should be read from the off-chip memory each time. However, if reusing the overlapped data, the $W \cdot 2 \cdot M$ i-tile should be read from the off-chip memory because the overlapped data are previously stored in the on-chip buffer. Consequently, the DSP only needs to read $W \cdot 2 \cdot M$ more data from the off-chip memory than the original ifmp data (WHM). However, if reuse is not performed, the DSP should read the ifmp data, which are approximately twice the size of the original ifmp data, from the off-chip memory.

IV. EVALUATION
A. SYSTEM CONFIGURATION

Fig. 11 depicts an integrated simulation environment for the real-time measurement of the energy consumption of the off-chip memory when a vector-SIMD DSP runs CNN applications. Cadence Vision P6 [27] is used as the vector-SIMD DSP, and the off-chip memory is modeled using the LPDDR4 memory [26]. The size of on-chip buffer is 256KB. For DSP simulation, XTSC (Xtensa SystemC) [28], which is a systemC-based simulator provided by Cadence, is used. In addition, DRAMSim [29], which is a cycle-accurate DRAM power simulator, is used for DDR power simulation. To integrate different simulation environments of DSP and DRAM, a systemC wrapper of DRAMSim is implemented. In the integrated simulation environment, DRAMSim acts as a systemC memory module in XTSC. The implemented wrapper includes a memory controller (MC) for XTSC, and systemC interface for the communication between XTSC and DRAMSim. The MC supports memory-access-format conversion, read before write, and reordering for in-order response.

First, the memory-access-format conversion overcomes the challenge of different memory access units of XTSC and DRAMSim. A request, memory access unit of P6, has the maximum transfer size of 256 bytes, whereas that of DRAMSim (transaction) is 64 bytes; therefore, format con-version is required. A request, memory access unit of P6, has the maximum transfer size of 256 bytes, whereas that of a transaction, the unit of DRAMSim, is 64 bytes. Therefore,
TABLE 2. Estimating the length of grouped layers according to the proposed energy-consumption model (Tiny YOLO v2).

| Conv1 | Pool1 | Conv2 | Pool2 | Trans | Conv3 | Pool3 | Conv4 | Total | $E_{\text{sim}}$ (mJ) | $E_{\text{model}}$ (mJ) |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------------|-----------------|
| Base  |       |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 1.967 | 0.648 | 2.362 | 0.341 | 0.148 | 1.448 | 0.19  | 1.152 | 0.117 | 8.373           | 1.94            |
| Power(P)  | 0.218 | 0.678 | 0.145 | 0.647 | 0.602 | 0.099 | 0.587 | 0.084 | 0.494 | -               | 0.449           |
| G1,3+   |       |       |       |       |       |       |       |       |     |                      |
| Base  |       |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 5.065 | 0.075 | 1.448 | 0.19  | 1.152 | 0.117 | 7.972 | -     | 0.79 | 0.265  |
| Power(P)  | 0.05  | 0.084 | 0.084 | 0.084 | 0.084 | 0.084 | 0.084 | -     | -    | -         |
| G1,4    |       |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 11.692 | 0.05  | 1.152 | 0.117 | 8.298 | -     | 0.51  | 0.197 | 0.245 | 0.184           |
| Power(P)  | 0.047 | -     | -     | -     | -     | -     | -     | -     | -    | -            |

(*) denotes ranking.

TABLE 3. Estimating the length of grouped layers according to the proposed energy-consumption model (VDSR).

| Conv1 | Trans | Conv2 | Conv3 | Conv4 | Conv5 | Conv6 | Total | $E_{\text{sim}}$ (mJ) | $E_{\text{model}}$ (mJ) |
|-------|-------|-------|-------|-------|-------|-------|-------|----------------|-----------------|
| Base  |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 5.460 | 3.648 | 36.028 | 36.028 | 36.028 | 36.028 | 189.248 | -     | 22.326 | 18.376 |
| Power(P)  | 0.201 | 0.722 | 0.103 | 0.103 | 0.103 | 0.103 | -     | -     | (6)   | (6)     |
| G1,3+   |       |       |       |       |       |       |       |     |                      |
| Base  |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 62.224 | 41.731 | 36.028 | 36.028 | 36.028 | 36.028 | 185.843 | -     | 16.199 | 14.9    |
| Power(P)  | 0.031 | 0.103 | 0.103 | 0.103 | 0.103 | 0.103 | -     | -     | (5)   | (5)     |
| G1,4    |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 52.246 | 91.451 | 36.028 | 36.028 | 36.028 | 36.028 | 199.535 | -     | 12.846 | 11.447 |
| Power(P)  | 0.018 | 0.103 | 0.103 | 0.103 | 0.103 | 0.103 | -     | -     | (4)   | (4)     |
| G1,4+   |       |       |       |       |       |       |       |     |                      |
| Base  |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 36.264 | 130.116 | 36.028 | 36.028 | 36.028 | 36.028 | 202.172 | -     | 9.492 | 8.006 |
| Power(P)  | 0.015 | 0.103 | 0.103 | 0.103 | 0.103 | 0.103 | -     | -     | (3)   | (3)     |
| G1,4    |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 26.246 | 203.358 | 608.928 | 608.928 | 608.928 | 608.928 | 239.386 | -     | 6.484 | 4.613 |
| Power(P)  | 0.013 | 0.103 | 0.103 | 0.103 | 0.103 | 0.103 | -     | -     | (2)   | (2)     |
| G1,4+   |       |       |       |       |       |       |       |     |                      |
| Base  |       |       |       |       |       |       |       |     |                      |
| Time(ns) | 18.184 | 16.164 | 14.164 | 12.164 | 10.164 | 8.164 | -     | -     | 5.146 | 2.41    |
| Power(P)  | 0.011 | -     | -     | -     | -     | -     | -     | -     | (1)   | (1)     |

the MC must convert the format between the two units for achieving correct simulation. In Fig. 11, the Request → Transaction module performs this conversion. For example, when the size of a request by the iDMA exceeds 64 bytes, the request is converted into multiple transactions; this scenario represents most of the cases in CNN applications, where memory access tends to be contiguous. In addition, the FIFO size of DRAMSim and MC, which are both equal to 32, is experimentally determined to prevent the stall due to FIFO fullness.

Another problem due to the different memory access format is partial write. In this study, partial write refers to the case in which the request size is smaller than the transaction size, i.e., 64 bytes. In the case of partial write, the validity of the remaining bytes not written by the request is not guaranteed. Therefore, partial write should be performed after reading the corresponding transaction from DRAM. In addition, when consecutive partial writes exist for the same transaction block, the remaining bytes of each write should bypass the pending writes that are previously requested to the DRAM. To counter this challenge, the proposed method implements read before write function to read the remaining bytes from the pending writes if any; otherwise, it reads from the DRAM.

In addition, P6 only accepts the in-order response to the request, whereas in the case of the DRAM, the order of the completed memory requests may be different from the order of the input according to the internal scheduling policy. Therefore, the proposed method implements a reorder buffer in the MC, then stores the request of P6 sent from iDMA in order, and finally responds in order of the reorder buffer after the DRAM operation is performed. Because the FIFO size of both the MC and DRAMSim is 32 and one transaction corresponds to the maximum number of 4 requests (16 bytes), the size of the reorder buffer is calculated to be 256 (= 64-4).
TABLE 4. Energy-consumption comparison for Tiny YOLOv2.

|                  | Energy consumption (mJ) | Execution time (ms) |
|------------------|-------------------------|---------------------|
|                  | Base                    | Base + inter-tile data reuse | Base + inter-layer data reuse | [21] | Proposed |
| Conv1            | 0.401 (1.945)           | 0.375 (1.911)        | 0.095 | 0.534 | 0.095 |
| Conv2            | 0.327 (2.347)           | 0.293 (2.344)        | (6.813) | (16.297) |
| Conv3            | 0.143 (1.435)           | 0.121 (1.434)        | 0.096 (1.140) | 0.096 (1.038) | 0.096 (1.140) |
| Conv4            | 0.096 (1.140)           | 0.074 (1.140)        | 0.113 (1.037) | 0.155 (1.038) | 0.113 (1.037) |
| Conv5            | 0.155 (1.038)           | 0.113 (1.037)        | 0.183 (1.237) | 0.183 (1.237) |
| Conv6            | 0.183 (1.237)           | 0.183 (1.237)        | 0.183 (1.237) | 0.183 (1.237) |
| Conv7            | 0.705 (6.795)           | 0.705 (6.795)        | 0.705 (6.795) | 0.705 (6.795) |
| Conv8            | 1.520 (7.228)           | 1.520 (7.228)        | 1.520 (7.228) | 1.520 (7.228) |
| Conv9            | 0.011 (0.050)           | 0.011 (0.049)        | 0.011 (0.050) | 0.011 (0.050) |
| Total (only 9 Conv) | 3.543 (23.218)          | 3.398                | - | - |
| Total            | 4.490 (24.808)          | 4.349                | 2.844 | 3.109 | 2.693 |
| Reduction (%)    | -                      | -                   | -3.14 | -36.65 | -30.75 | -40.02 |

TABLE 5. Energy-consumption comparison for VDSR.

|                  | Energy consumption (mJ) | Execution time (ms) |
|------------------|-------------------------|---------------------|
|                  | Base                    | Base + inter-tile data reuse | Base + inter-layer data reuse | [21] | Proposed |
| Conv1            | 1.094 (5.441)           | 1.094 (5.441)        | - | - |
| Conv2            | 3.717 (36.019)          | 2.395 (34.315)       | 7.205 | 7.205 | 7.205 |
| Conv3            | 3.717 (36.019)          | 2.395 (34.315)       | (480.337) | (480.337) |
| Conv4            | 3.717 (36.019)          | 2.395 (34.315)       | (480.337) | (480.337) |
| Conv5            | 3.717 (36.019)          | 2.395 (34.315)       | - | - |
| Conv6            | 3.717 (36.019)          | 2.395 (34.315)       | - | - |
| Total (only 20 Conv) | 71.721 (23.218)         | 46.604               | - | - |
| Total            | 74.336 (693.422)        | 49.219               | 22.889 | 22.889 | 22.889 |
| Reduction (%)    | -                      | -                   | -33.78 | -69.20 | -69.20 |

B. BASELINE AND BENCHMARK

A dataflow-scheduling method generated using the Xtensa Neural Network Compiler (XNNC) [30], which is an automated tool provided by Cadence, is used as the baseline. XNNC converts a floating-point CNN to a fixed-point DSP code by considering the target DSP hardware. For example, the tiling factor is determined by considering the size of the on-chip buffer (256KB), and the convolution kernel with high MAC utilization is selected by considering the channel depth of the output feature map. The bit-width is fixed to 8-bit. Consequently, the DSP code generated by XNNC achieves a similar performance to that of the original floating-point CNN, although the iterative quantization process is employed. However, the baseline offers limitation minimization of the off-chip memory access because the dataflow and tiling are determined without considering the characteristic of the given CNN. This is because in the baseline, the dataflow used is the layer-wise dataflow, and the tiling for the W-axis is not considered.

The proposed method and baseline are compared with each other on the basis of their performances in three CNN applications, namely, Tiny YOLOv2 [31], VDSR [32], and MobileNetv1 [34]. Reference [31] presents a widely used, high-quality object-detection network, which comprises nine convolutional layers and six max pooling layers and enables real-time operation even in embedded systems. Reference [32] is a VGG [33] based image-enhancement network consisting of 20 convolutional layers, where the configuration of
all the layers is the same except for the first one. Reference [34] is particularly useful for the mobile and embedded systems because it has small model size and small computational complexity. Reference [33] consists of 27 convolutional layers (1 convolutional layer, 13 depth-wise convolutional layer, 13 point-wise convolutional layer) and the size of input image is all $416 \times 416 \times 3$.

**C. OPTIMAL LENGTH OF GROUPED LAYERS**

The length of the grouped layers should be determined before the DSP runs the target CNN application. To that end, this paper proposes (1), using which this paper estimates the energy consumption of the off-chip memory.

Table 2 and 3 present $E_{\text{sim}}$ and $E_{\text{model}}$ for Tiny YOLOv2 and VDSR, respectively. $E_{\text{sim}}$ and $E_{\text{model}}$ denote the energy consumption measured using the memory simulator DRAMSim and (1), respectively. The tile size is determined by the optimization algorithm (5) before calculating (1). In addition, $\text{Conv}$, $\text{Pool}$, and $\text{Trans}$ denote the convolutional layer, pooling layer, and the transpose layer that converts the structure of the feature map for applying the optimal convolution kernel, respectively. Base denotes the dataflow scheduling via XNNC. $G_{i-2}$ denotes grouping the layers from $\text{Conv}_i$ to $\text{Conv}_{i-2}$, and $G_{i-1}$, $G_{i-4}$, $G_{i-5}$, and $G_{i-6}$ are defined equally. The layers from $\text{Conv}_1$ to before $\text{Conv}_5$ and $\text{Conv}_1$ to before $\text{Conv}_7$ cannot be grouped for Tiny YOLOv2 and VDSR, respectively, because $\text{Wght}_{i-5}$ and $\text{Wght}_{i-7}$ exceed the size of the on-chip buffer. $E_{\text{model}}$ is calculated in only $\text{Conv}$ except for $\text{Pool}$ and $\text{Trans}$. Time denotes the execution time (in milliseconds (ms)) when the operating frequency of the DSP is 1 GHz. Power denotes the average power (in watt (W)). Therefore, the unit for $E$ is mJ (ms $\cdot$ W).

Basically, it can be expected that $\text{Time}$ increases as the length of the grouped layers increases, however there is...
exception: $G_{1-2}$+Base and $G_{1-3}$+Base for Tiny YOLOv2, $G_{1-2}$+Base for VDSR. The original layer-wise dataflow requires the preparation time for the local memory initialization and the first i-tile data transfer from the off-chip memory to the on-chip buffer per each layer. However, this preparation time is skipped in the grouped layers. In the above exception cases, the skipped preparation time is larger than the processing time increased by the overlapped SIMD operations.

$E_{sim}$ decreases until the length of grouped layers reaches the optimal value. According to $E_{sim}$, the optimal length of the grouped layers is 3 for Tiny YOLOv2 and 6 for VDSR when the initial layer of the grouped layers is $Conv_1$; the considerable difference between the optimal lengths of Tiny YOLOv2 and VDSR can be attributed to the pooling layer. For Tiny YOLOv2, the amount of data transfer between the DSP and the off-chip memory is reduced by increasing the length of the grouped layers. However, for VDSR, the same amount of data transfer is retained despite increasing the length of the grouped layers. Therefore, in terms of $E_{sim}$, it is not optimal for Tiny YOLOv2 to maximize the length of the grouped layers.

Although $E_{model}$ is different from $E_{sim}$, the ranking of the former is same as that of the latter. There are three reasons for this. The first reason is that the activation/precharge energy and refresh energy, both of which the $E_{model}$ does not consider, comprise only a small portion of the total energy consumption, as the address of the data requests by P6 to the off-chip memory is contiguous. In addition, the refresh energy is a constant consumption of the off-chip memory and, therefore, does not affect ranking. The second reason is that the size of the data that the P6 request to the off-chip memory is different from the size of the data that are processed using the off-chip memory. The processing unit of the off-chip memory is a transaction, which processes 64 bytes of data at once. The smaller is the size of the data requested by P6 less than 64 bytes, the larger is $E_{sim}$ compared to $E_{model}$. However, this difference does not affect the ranking because it increases proportionally to the size of the data that are requested by P6.

The third reason is that $E_{model}$ does not consider the energy consumption for the pooling and transpose layers. Because $Time$ for both these layers is very small compared with that for the convolutional layer, the ranking of $E_{model}$ is not affected. The following shows the calculation of the $G_{1-2}$’s $E_{model}$ using (1) in Tiny YOLOv2:

$$E_{1-2} = (P_{act} \cdot T_{act} + P_{idle} \cdot T_{idle}) \cdot N_{Tile}$$

$$= (P_{act} \cdot (F_{1-2} \cdot Mem_{Op}/Mem_{Freq}) + P_{idle} \cdot (Grouped_{N_{SIMD}}/DSP_{Freq} - T_{act})) \cdot N_{Tile}$$

$$= ((0.801/504) \cdot (234, 048 \cdot 16/(1.5 \cdot 10^9))$$

$$+(0.0016) \cdot (22, 659, 072/10^9 - (234, 048 \cdot 16/(1.5 \cdot 10^9)))$$

$$-64$$

where $P_{act}$ and $P_{idle}$ denote (IDD4R/IDD4W). VDD and IDD2P. VDD, respectively. In addition, IDD4R, IDD4W, IDD2P are equal to 458.18, 728.18, and 1.45 mA, respectively. VDD and Mem_{Op} are equal to 1.1 V and 16 clock cycle, respectively. These values are referred to as the memory datasheet [26]. $Grouped_{N_{SIMD}}, F_{1-2}$, and $N_{Tile}$ are 22,659,072, 234,048, and 64, respectively. These values depend on the optimal tile size. Mem_{Freq} and DSP_{Freq} are equal to 1.5 and 1.0 GHz, respectively.

As shown in the above-mentioned calculation, the optimal length of the grouped layers depends on the memory model used and on how much computing resource such as the MAC array is supported. In addition, the selected length optimizes only the energy consumption of the off-chip memory and not the processing speed. Therefore, for optimizing the processing speed, a length smaller than the selected length might be considered.

The GL for Tiny YOLOv2 and VDSR are $\{3,1,1,1,1,1\}$ and $\{6,6,6,2\}$, respectively. For Tiny YOLOv2, other layers cannot be grouped because the sum of weight data exceed the size of the on-chip buffer per layer.

D. COMPARISON OF ENERGY CONSUMPTION

This section verifies the energy-consumption reduction and execution time achieved via each proposed method and the original layer fusion method [21], following which this section compares them with the energy-consumption reduction and the execution time achieved via the baseline method.

Table 4, 5, and 6 list the energy consumption of each method for the convolutional layers of Tiny YOLOv2, VDSR, and MobileNetv1, respectively. Base + inter-tile data reuse and Base + inter-layer data reuse methods denote that only the selection of the optimal tile size for each single layer is applied to the baseline, and that only layer grouping is applied. Reference [21] uses that the layers are grouped as much as possible from the first layer. Proposed denotes the method that applies both reuse schemes. The inter-layer data reuse is applied first, following which inter-tile data reuse is applied to the layers that cannot be grouped. Reduction denotes the amount of energy reduction achieved using each method compared with the total energy consumption achieved via the baseline.

The reductions achieved using Base + inter-tile data reuse are $-3.14\%, -33.78\%$, and $-8.36\%$ for Tiny YOLOv2, VDSR, and MobileNetv1, respectively. In Tiny YOLOv2, no data transfer occurred from $Conv_6$ to $Conv_9$ because the feature map fits in the on-chip buffer from $Conv_6$ to $Conv_9$, and in MobileNetv1, inter-tile data reuse cannot be applied in the point-wise convolutional layer because the filter size is $1 \times 1$. However, VDSR does not. Thus, inter-tile data reuse is more effective in VDSR than Tiny YOLOv2 and MobileNetv1. The execution time is all reduced after applying inter-tile data reuse since the latency for the off-chip memory transfer is reduced.

In addition, the reductions using Base + inter-layer data reuse are $-36.65\%, -69.20\%$, and $-41.66\%$ for
Tiny YOLOv2, VDSR, and MobileNetv1, respectively. In VDSR, all the layers can be grouped separately, whereas in Tiny YOLOv2 and Mo-bileNetv1 the same are not possible. Thus, the reduction achieved using VDSR is greater than that achieved using Tiny YOLOv2 and MobileNetv1, and there is no difference between the proposed method and the baseline. In terms of the execution time, [21] shows the execution time is all increased. However, Proposed shows the execution time does not change significantly compared to the baseline even though the additional computation happened by the layer grouping, except for VDSR. This is because [21] performs the layer grouping until the grouping is possible without considering the increased computational complexity, whereas the Proposed searches the optimal group length considering the computational complexity. As the results, the proposed method can minimize the execution time for the additional computation by the layer grouping. Through the simulation results, this optimal point is right after the pooling layer or the convolutional layer with stride 2. In VDSR, the execution time significantly increases as in [21] even if the proposed method is applied. The reason is that all layers are same, therefore there is no optimal point.

V. CONCLUSION

In this study, a dataflow-scheduling scheme, which minimizes the access to the off-chip memory, was presented for the DSP systems. In the proposed dataflow scheduling, the given CNN was grouped separately; furthermore, the optimal length of each grouped layer was estimated on the basis of the length that resulted in the minimum estimated energy consumption of the off-chip memory. In addition, the optimal tile size was determined for all the convolutional layers. For the grouped layers, the tile size was determined to minimize the overlapped computation. However, for the non-grouped layers, the tile size was determined to maximize the reuse of the overlapped data be-tween successive tiles. Consequently, using the proposed method, an energy-consumption reduction of 51% could be achieved in an integrated simulation environment compared to the baseline.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.
[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2014, pp. 580–587.
[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated recognition, localization and detection using convolutional networks,” 2013, arXiv:1312.6229.
[5] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 5, pp. 898–916, May 2011.
[6] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A deep convolutional encoder–decoder architecture for image segmentation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 12, pp. 2481–2495, Dec. 2017.
[7] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” ACM Sigplan Notices, vol. 42, no. 1, pp. 269–284, 2014.
[8] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, “Cembricon-X: An accelerator for sparse neural networks,” in Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2016, pp. 1–12.
[9] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durandovic, E. Cosatto, and H. P. Graf, “A massively parallel coprocessor for convolutional neural networks,” in Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors, Jul. 2009, pp. 53–60.
[10] S. Charkradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, “A dynamically reconfigurable coprocessor for convolutional neural networks,” ACM SIGARCH Comput. Archit. News, vol. 38, no. 3, pp. 247–257, 2010.
[11] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, “Memory-centric accelerator design for convolutional neural networks,” in Proc. IEEE 31st Int. Conf. Comput. Design (ICCD), Oct. 2013, pp. 13–19.
[12] Q. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, Sun, and Y. Guo, “And J. Cong, Sun,” High performance vector-SIMD DSP,” in Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors, Jul. 2009, pp. 53–60.
[13] T. Chen, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, “Cembricon-X: An accelerator for sparse neural networks,” in Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2016, pp. 1–12.
[14] Z. Du, R. Fasthuber, T. Chen, P. Lenne, L. Li, T. Luo, and O. Temam, “ShiDianNao: Shifting vision processing closer to the sensor,” ACM SIGARCH Comput. Archit. News, vol. 43, no. 3S, pp. 92–104, 2015.
[15] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, “A 46.6 A1.93TOPS/W scalable deep learning/inference processor with hexa-parallel MIMD architecture for big-data applications,” in IEEE Int. Solid-State Circuits Conf. (ISSCC), Dig. Tech. Papers, Feb. 2015, pp. 1–3.
[16] J. Ngiam, Z. Chen, D. Chia, P. V. Koh, Q. V. Le, and A. Y. Ng, “Tiled convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2010, pp. 1279–1287.
[17] A. Stiouchkin, F. Conti, and L. Benini, “Optimally scheduling CNN convolutions for efficient memory access,” 2019, arXiv:1902.01492.
[18] G. Erland, S. Parikh, H. Sanghavi, and A. Farouqui, “High performance DSP for vision, imaging and neural networks,” in Proc. IEEE Hot Chips Symp. (HCS), Aug. 2016, pp. 1–30.
[19] C. Yang, S. Chen, Y. Wang, and J. Zhang, “The evaluation of DCNN on vector-SIMD DSP,” IEEE Access, vol. 7, pp. 22301–22309, 2019.
[20] M. Alwani, H. Chen, M. Ferdman, and P. Mildor, “Fused-layer CNN accelerators,” in Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2016, pp. 1–12.
[21] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, and C. Guestrin, “TVM: An automated end-to-end optimizing compiler for deep learning,” in Proc. 13th USENIX Symp. Operating Syst. Design Implement., 2018, pp. 578–594.
[22] H. Kwon, P. Chatarasi, P. Pellauer, M. Parashar, A. V. Sarkar, and T. Krishna, “Understanding re-use, performance, and hardware cost of dnn dataflows: A data-centric approach,” in Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture, Oct. 2019, pp. 754–768.
[23] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, “Collaborative execution of deep neural networks on Internet of Things devices,” 2019, arXiv:1901.02537.
[24] L. Zhou, H. Wen, R. Teodorescu, and D. H. Du, “Distributing deep neural networks with containerized partitions at the edge,” in Proc. Workshop Hot Topics Edge Comput., 2019, pp. 1–7.
[25] Mobile LPDDR4 SDRAM, Micron, Boise, ID, USA, 2015.
[27] Vision P6 DSP User’s Guide, Cadence Design Systems, San Jose, CA, USA, 2016.

[28] Xtensa SystemC, Cadence Design Systems, San Jose, CA, USA, 2018.

[29] P. Rosenfeld, E. CooperBalis, and B. Jacob, “DRAMSim2: A cycle accurate memory system simulator,” IEEE Comput. Archit. Lett., vol. 10, no. 1, pp. 16–19, Jan./Jun. 2011.

[30] Xtensa Neural Network Compiler, Cadence Design Systems, San Jose, CA, USA, 2018.

[31] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 7263–7271.

[32] J. Kim, J. K. Lee, and K. M. Lee, “Accurate image super-resolution using very deep convolutional networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 1646–1654.

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

[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.

WONTAE KIM received the B.S. and M.S. degrees from the School of Electronics, Telecommunication and Computer Engineering and Information Engineering, Korea Aerospace University, Gyeonggi, South Korea, in 2014 and 2016, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University. His research interests include the algorithm and architecture for computer vision and multimedia applications.

SANGHEON LEE received the B.S. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2017, where he is currently pursuing the integrated M.S. and Ph.D. degree in electrical and computer engineering. His research interests include learning algorithms and VLSI architecture for computer vision.

ILWI YUN received the B.S. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2018, where he is currently pursuing the integrated M.S. and Ph.D. degree in electrical and computer engineering. His research interests include deep learning and VLSI architecture for computer vision.

CHULHEE LEE received the B.S. degree in electronics and computer engineering from Hanyang University, Seoul, South Korea, in 2012, and the Ph.D. degree in electrical and computer engineering from Seoul National University, Seoul, in 2018. He is currently a Staff Engineer at the System LSI Division, Samsung Electronics Corporation, Hwasung, South Korea. His research interests include GPU parallelization and VLSI architecture for computer vision and multimedia applications.

KYUJOONG LEE (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2002, the M.S. degree in electrical engineering from the University of Southern California, Los Angeles, USA, in 2008, and the Ph.D. degree in electrical engineering and computer science from Seoul National University, in 2013. From 2002 to 2005, he was with Com2us Corporation, Seoul, as a Developer. From 2013 to 2017, he has worked at the S.LSI Division, Samsung Electronics Corporation. In 2017, he was appointed as an Assistant Professor with the Department of Electronic Engineering, Sun Moon University, Asan, South Korea. In 2022, he moved as an Associate Professor to the School of AI Convergence, Sungshin Women’s University. His major research interests include the algorithms and architectures of deep learning and image/video compression and processing.

HYUK-JAE LEE (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1996. From 1998 to 2001, he was a Senior Component Design Engineer at the Server and Workstation Chipset Division, Intel Corporation, Hillsboro, OR, USA. From 1996 to 1998, he was a Faculty Member at the Department of Computer Science, Louisiana Tech University, Ruston, LA, USA. In 2001, he joined the School of Electrical Engineering and Computer Science, Seoul National University, where he is currently a Professor. He is the Founder of Mamurian Design Inc., Seoul, a fabless SoC design house for multimedia applications. His current research interests include computer architecture and SoC for multimedia applications.

VOLUME 10, 2022