Received 3 August 2022, accepted 18 August 2022, date of publication 5 September 2022, date of current version 15 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204038

RESEARCH ARTICLE

A Compressed Data Partition and Loop Scheduling Scheme for Neural Networks

DEJIAN LI[1], RONGQIANG FANG[2], JING WANG[3], DONGYAN ZHAO[1], TING CHONG[1], ZENGMIN REN[1], AND JUN MA[1]
[1]Beijing Smartchip Microelectronics Technology Company Ltd., Beijing 100192, China
[2]School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
[3]School of Information, Renmin University of China, Beijing 100056, China

Corresponding author: Jing Wang (jwang@ruc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62076168, and in part by the Laboratory Open Fund of Beijing Smart-Chip Microelectronics Technology Company Ltd.

ABSTRACT Neural networks (NNs) have been widely adopted in various application domains. Deeper NNs greatly enhance the output accuracy, but complex NNs with more parameters incur intensive memory accesses, and the data usually need to be partitioned since it may exceed the on-chip storage. However, there is no research considering the partition and scheduling co-design of the NNs. In this paper, we propose a sparse NN data partition and loop scheduling scheme. We establish the compression efficiency model of the matrix sparse algorithm and design a partition selection method based on sparsity characteristics analyzed by the compression efficiency model. Further, we design a loop scheduling scheme based on the proper partition size. The experiment results show that the average memory access of each layer can be compressed to 68% of the original, and the throughput of the AlexNet, VGG and VGG19 is increased to an average of 1.66 times.

INDEX TERMS Sparse matrix, neural networks, loop scheduling, compression, partition.

I. INTRODUCTION

With the prosperity and development of the mobile internet, artificial intelligence applications based on deep neural networks are gradually migrating from cloud computing to mobile computing. With its wide application in image procession and speech recognition, the complexity of deep learning algorithms continues to increase to meet the requirements of modern applications. However, the large number of computations and memory accesses in deep neural networks generates considerable energy. There is a large contradiction between the power consumption and the limited power supply capability of mobile devices. In addition, large-scale network model data need to be transmitted to devices with limited storage. When the storage capacity is smaller than the model parameter size, the data usually need to be partitioned, thus, how to achieve optimal data partitioning and transformation to ensure efficient data processing becomes a hot research topic [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Ching Ying.

Since the weights of even a single convolutional layer can exceed the local storage capacity, researchers proposed data transformation schemes: graph partitioning uses the synchronous dataflow model, which splits the graph into subgraphs along convolutional layers and maps each subgraph to a different bitstream, however, this scheme requires FPGA (Field-Programmable Gate Arrays) reconfiguration when data flows to the next subgraph.

Folding is also an effective method to partition the data: this kind of method folds input by a factor, and a convolutional layer is split into multiple subgraphs that execute a fraction of total convolutional. The interim results are accumulated to generate the output. Thus, the storage requirement is reduced by the folding factor. The folding methods are further divided into coarse-grain folding and fine-grain folding. Coarse-grain folding fully unrolls the major operations of every layer and provides the highest throughput possible. Fine-grain folding is a time-multiplexed scheme between different operations, which use much smaller numbers of hardware units.

In addition, the parameter matrix of a neural network is usually sparse, and compression can effectively reduce the storage space requirements. However, there is no research considering the partitioning of the spare matrix, which compression algorithm is selected, and how partitioning and scheduling the matrix computation directly affect the performance. Therefore, this paper focuses on how to select the compression scheme and partition data based on the matrix sparsity of different layers in the convolutional neural networks and proposes a sparsity model and partition scheme for the neural network. The main contributions of this paper are as follows:

1. We established the compression efficiency model of the matrix sparse algorithm, proposed the sparse coefficients and sparse offsets of different algorithms, and calculated the compression ratio of different compression algorithms.
2. We analyzed the sparsity characteristics of the convolutional neural network parameter matrix, introduced the concept of average density, and predicted the amount of sparse matrix memory access through statistical information to provide guidance for partitioning strategies.
3. We designed a partition size selection method for sparse neural networks. We first analyzed the relationship between memory access and model scale/sparsity, and then used the dynamic programming algorithm to solve the partition size selection problem.
4. We designed a loop scheduling scheme based on the proper partition size, and the optimal mapping method was designed for the sparse neural network.

II. RELATED WORKS

A. SPARE MATRIX COMPRESSION

Previous research has shown that most of the parameter matrices of neural networks are sparse, and compressed sparse matrices can effectively reduce memory overhead. Typical compression schemes include COO, CSR, SCNN, and Swallow.

We use sparse coefficients and sparse offsets to characterize the compression effect of the compression algorithm. The sparse coefficient α is used to characterize the memory resources occupied by each nonzero value in the actual storage requirements. The coefficient is only changed according to the sparse algorithm rules, and dose not vary with different matrices, the sparse offset β describes index size or data location, which is related to the layout of the nonzero matrix values.

COO [2] uses row number, column number, and element value to store nonzero elements. For nonzero elements in any matrix, three values need to be recorded, so its sparse coefficient is three, which means that when the number of nonzero elements is K, the storage space overhead is 3K. COO is a triple storage algorithm, that is effective when the matrix is very sparse, and it is less efficient for dense matrix compression.

The CSR scheme stores data values, column index, and row index. The row index records the number of newlines between adjacent data, that is, if there is no newline between adjacent nonzero data, the newline index will not be added. The sparse coefficient of the CSR algorithm is two. Ideally, only values and column numbers are stored, and the storage space overhead is 2K. However, the actual situation cannot guarantee that all nonzero values are in the same row, nor can it guarantee that all data are evenly distributed in each row. Therefore, the CSR algorithm requires 2K+c storage space, where c is the total number of rows occupied by data.

In addition to storing numerical bits, SCNN [3] also needs to store the number of 0 values from each data to the previous nonzero data. An additional data bit is stored to represent the number of nonzero values in the matrix. Therefore, the sparse coefficient of SCNN is two, and the storage requirement is 2K+1.

The Swallow [4] scheme records the number of nonzero values in the same row for each channel and uses the offset bit to record the column number of the data. The sparse coefficient of the Swallow scheme is two. The sparse offset is R, which is the number of rows.

B. LOOP SCHEDULING

Scheduling research on CNN(Convolutional Neural Networks) accelerators has a long history. Since the 1980s, a series of as early/late as possible algorithms [5], degrees of freedom-based algorithms [6], and mobility-based scheduling algorithms have been proposed [7], [8], [9], [10], [11]. Then, researchers found that the scheduling problem is not only how to split the operator into each multiplier-accumulator, but also how to implement a multi-batch pipeline for different convolutional layers [12]. Therefore, optimization is carried out from two aspects: resource allocation and dataflow [13]: resource allocation methods mainly focus on how to improve resource utilization, while data flow methods solve how to transport data to achieve the highest performance.

In terms of resource allocation optimization, the ACT laboratory extracts the the control components hyperparameters from the perspective of the Process Element(PE) configuration for optimization, imitating the Von Neumann control mode, with the help of a large number of expertly pre-defined template optimization to add instructions to guide dynamic scheduling in the FPGA [5]. [14] compared the size of the convolution kernel matrix with the memory bandwidth, thereby providing the basis for data partitioning and solving the problem of uneven resource allocation. Loop tiling and loop unrolling consider both on-chip data reuse and external storage and can be used to improve mapping efficiency and resource utilization [15]. [12] noted that the scheme of providing convolution processing unit for each stage of the convolution computation [16] is inefficient, and therefore proposed multiple multiprocessing structures with different computing capabilities to pipeline the processing. [17] designed a scheduling algorithm based on the maximum value to improve the resource utilization problem under multi process element cooperative computing. A reloadable architecture of neural network was proposed in [18]. The above

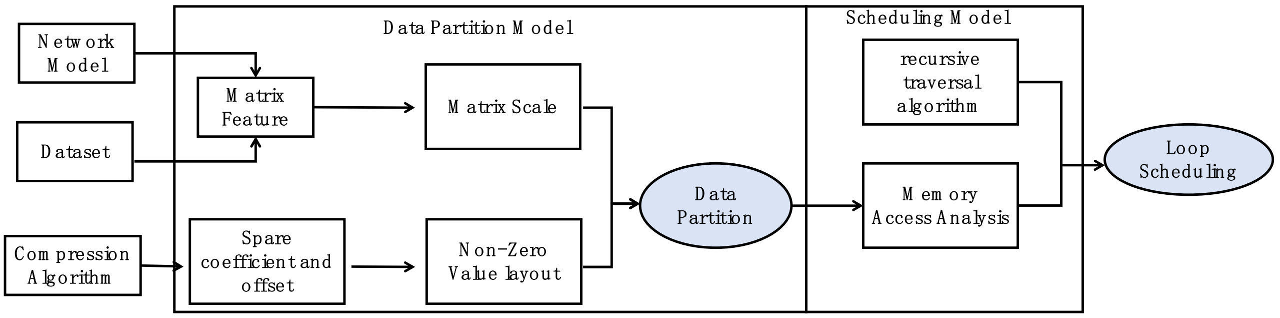

FIGURE 1. **Flow of the data partition and loop scheduling scheme.**

scheme is simple and direct, but it is only effective on specific operators, and the generality is poor.

The data flow optimization problem involves modifying the control logic or timing relationship of the hardware, scheduling the flow order of the data, maximizing the reuse rate and enhancing the parallelism of data transmission/processing. [15] designed three on-chip buffer structures: full reuse, partial reuse, and independent handling. Arthur Stoutchinin *et al.* proposed a model HWC [19] to efficiently evaluate the reusability of network mappings, and based on the model, a better data scheduling scheme was given. Li *et al.* proposed a layer partition scheduling strategy SmartShuttle based on the difference in sensitivity of input data, input data and weight to reusing different convolutions [20]. Alwani *et al.* proposed caching the area shared by adjacent receptive fields, so that only the nonshared part needs to be fetched from memory in the next layer calculation [21].

Previous works focused on how data are transferred to the on-chip buffer, and [22] started with the data that have been transferred to the chip. Distributing the convolution operation and the fully connected operation on a 4-level parallelism design enables different types of network layers to make full use of the same multiply-add operation unit to form a timing relationship that does not generate pipeline bubbles [22]. [23] changed the hardware storage method to two imensional data storage in the form of window size, so that the start time of the convolution operation is advanced to when the matrix storage meets the window size, thereby shortening the time for pipeline task completion [23]. Lu *et al.* proposed FlexFlow, a flexible dataflow accelerator architecture for convolutional neural networks [24], and divided the loop unrolling methods into three categories: unrolled convolution kernel matrix [25], [26], [27], unwrapped feature map matrix [16], [28], [29] and unwrapped matrix channels [30].

The Imperial College of London proposed a real-time reconfigurable architecture. The directed graph corresponding to the convolutional network is divided into several subgraphs, and each subgraph corresponds to a different hardware architecture. Bitstream files need to be reinjected between network layers belonging to different subgraphs to reconfigure the hardware architecture [31]. Kwon *et al.* designed a NN accelerator with a built-in set of reconfigurable building blocks that can easily support countless DNN partitions and maps by appropriately configuring the interconnect architecture [32]. Xu *et al.* proposed converting a neural network's hardware mapping into a dataflow graph. The graph shows whether the data to be processed by this layer is given by the output of the network model of the previous layer, so as to formulate the principle of time-sharing and multiplexing for PE(Processes Element) [33]. Rhu *et al.* [34] improved hardware performance through weight prefetching. Li proposed splitting the network to dynamically release memory resources [35]. Wicaksana Putra reused memory storage by filling data in any location that the memory can carry data, to optimize the data flow and improve the system performance [36].

Addressing scheduling problems can greatly optimize the model mapping so that both computing and storage resources can fully utilize their capacity. However the existing scheduling optimization schemes lack generality, and most scheduling solutions can only optimize some specific operators. Therefore, it is necessary to find a general solution to solve various neural networks consisting of complex operators and their variants [24], [37].

III. NNS' PARTITION AND SCHEDULING SCHEME

The amount of data transmission in the NNs is the key to optimizing performance. To determine the actual quantity of data to be transported, it is necessary to obtain the matrix size, density, and sparse algorithm features. Therefore, we analyze the storage characteristics of the matrix according to the network model and dataset and find the optimal compression storage scheme by comparing the original storage capacity and the sparsely compressed data storage capacity. After the compression scheme is determined, the memory access footprint of the model is calculated, and the partition size with the largest throughput is selected according to the footprint. Finally, the enumeration method is used to select the loop scheduling scheme.

A. MODELING OF SPARSE MATRIX STORAGE FEATURES

The feature analysis module calculates the storage g of the compressed matrix, compares the value of G_{comp} with the original uncompressed storage amount G_{orig}, and selects the strategy that occupies the smallest storage space. G_{comp} is a linear function of the number of nonzero

TABLE 1. Matrix parameter features.

parameters	
N/M	The number of input/output channel
R_I/C_I	The number of row/column of input
K	Size of the CNN kernel
R_O/C_O	The number of row/column of output
S	The strip of the network

TABLE 2. Feature of the sparse compression algorithm.

algorithm	COO	CSR	SCNN	Swallow
coefficient (α)	3	2	2	2
offset(β_i)	0	c	1	R
capacity(g)	$3\bar{k}$	$2\bar{k}+c$	$2\bar{k}+NM$	$2\bar{x}+R$

elements K, and the coefficient α and offset β are related to the sparse matrix compression algorithm. The number K of nonzero values in the matrix varies with the model and input data, so the average number of nonzero values $\bar{k}$ is introduced. To describe the sparsity of the matrix, the average density $\bar{D}$ is introduced, which is the normalization of the number of nonzero values. To calculate $\bar{k}$ and $\bar{D}$, the feature parameters of the matrix are extracted, as shown in Table 1. Note that the number of nonzero values in the input/weight/output matrix is $k_{IFM}, k_{KER}, k_{OFM}$, and the original storage requirements are $G_{IFM}, G_{KER}, G_{OFM}$. Then the average density of different matrices can be calculated using Formulation (1)-(3).

$$G_{IFM}=R_I\times C_I, \bar{D}_{IFM}=\frac{k_{IFM}}{N\times G_{IFM}} \quad (1\text{-}1)$$

$$\bar{k}_{IFM}=N\times G_{IFM}\times \bar{D}_{IFM} \quad (1\text{-}2)$$

$$G_{KER}=K^2, \bar{D}_{KER}=\frac{k_{KER}}{N\times M\times G_{KER}} \quad (2\text{-}1)$$

$$\bar{k}_{KER}=N\times M\times G_{KER}\times \bar{D}_{KER} \quad (2\text{-}2)$$

$$G_{OFM}=R_O\times C_O, \bar{D}_{OFM}=\frac{k_{OFM}}{M\times G_{OFM}} \quad (3\text{-}1)$$

$$\bar{k}_{OFM}=M\times G_{OFM}\times \bar{D}_{OFM} \quad (3\text{-}2)$$

The matrix feature extraction module obtains the matrix size of the input/weight/output matrix, the number of nonzero values and the layout of nonzero values. We use the feature information of the matrix, and the sparse coefficient and offset of the compression scheme to calculate the storage capacity of the compressed sparse matrix, as shown in Formulation (4).

$$g=\bar{k}\alpha+\sum_{i=0}^{\bar{k}}\beta_i \quad (4)$$

Based on the above formula, the storage capacity of a typical sparse matrix compression algorithm can be calculated, as shown in Table 2.

The layers close to the input layer have more original information, and the corresponding matrix is denser. The layers close to the output layer are sparser because the network removes redundant information and retains core features.

TABLE 3. Parameters of the data transfer calculation.

	λ	μ
$Input$	$\left\lceil\frac{M}{P_{out}}\right\rceil\times\frac{N}{P_{in}}$	$P_{in}\times g_{IFM_0}$
$Kernel$	$\left\lceil\frac{M}{P_{in}}\right\rceil\times\frac{N}{P_{in}}$	$P_{out}\times P_{in}\times g_{KER_0}$
$Output$	$\frac{M}{P_{out}}$	$P_{out}\times g_{OFM_0}$

Therefore, for a denser input-side matrix, the compressed storage cost is relatively high. Therefore, it is necessary to judge whether the compressed storage amount will exceed the original size, that is, $G_{comp} > G_{oirg}$. If the above situation occurs, the original storage without compression is selected.

B. THROUGHPUT-BASED PARTITION SIZE SELECTION

The size of the neural network model usually exceeds the on-chip storage, and the matrix partitioning technique is needed to transfer data to FPGA. Ideally, the partitioned input matrixT, output matrix, and weight matrix are stored in the on-chip buffer until the partial sum is completely superimposed and then written back to the memory. The total execution latency includes memory access time and computation time. The memory access time is the product of the bit transmission delay and the amount of transmitted data. The quantity of transmitted data is the sum of the input, weight, and output matrix transmissions. The amount of each matrix X transmission is the matrix transmissions λ_X and the single transmission data amount μ_X, as shown in Table 3. P_{in} is the partition size of input and Pout is the partition size of output, and Our propose is finding the best partition size P_{in} *and* P_{out}

The total transmission volume is calculated as shown in Equation (5):

$$Total\ Data\ Transfer\ (TDT)=\lambda_I\mu_I+\lambda_K\mu_K+\lambda_O\mu_O \quad (5)$$

The calculation of the memory access time MAT is shown in Equation (6), where A is the transmission time of single-bit data, which is only related to the system bit width and bandwidth.

$$Memory\ Access\ Time\ (MAT)=A\times TDT \quad (6)$$

The computing time is obtained by multiplying the total computing partitions and the computing time of each partition, where the computing time of each partition is represented by the data size γ divided by the frequency f, and γ is obtained by dividing the total network parameter size by the memory bandwidth B, as shown in Equation (7)

$$\gamma=\left\lceil\frac{g_{OFM_0}\times g_{KER_0}}{B}\right\rceil \quad (7)$$

Therefore, the final computation time is expressed as Equation (8):

$$Total\ Computation\ Time\ (TCT)=\left\lceil\frac{M}{P_{out}}\right\rceil\times\left\lceil\frac{N}{P_{in}}\right\rceil\times\frac{\gamma}{f} \quad (8)$$

Throughput is the total amount of transferred data divide the latency. The transmission and computation of data are often performed in parallel, so latency is the muximum of TCT and MAT, then throughput is shown in Equation (9).

$$Throughput = \frac{2 \times N \times M \times g_{OFM_0} \times g_{KER_0}}{Max\{TCT, MAT\}} \tag{9}$$

Partition selection is a multi-objective optimization problem that needs to maximize throughput while minimizing memory occupation. The constraints are that the partition size is smaller than the channels of the neural network model and the memory capacity *Capacity*, which is shown in Equation (10).

$$constriction \begin{cases} 1 \le P_{in} \le N & (10-1) \\ 1 \le P_{out} \le M & (10-2) \\ \mu_I + \mu_K + \mu_O \le Capacity & (10-3) \end{cases} \tag{10}$$

The objective function is shown in Equation (11).

$$goal \begin{cases} min(Capacity - (\mu_I + \mu_K + \mu_O)) \\ max(Throughput) \end{cases} \tag{11}$$

Our propose is finding the partition size P_{in} *and* P_{out}, and we propose a throughput-based maximum partition selection algorithm. The algorithm transforms the above dual-objective optimization problem into a single-objective and multiple constraint optimization problem. The algorithm first sets MAT = TCT, uses P_{in} to represent P_{out} according to Equation 10-3, and then the range of Our propose is finding the partition size P_{in} can be represented by formulation (12)

$$\frac{N\gamma}{Af\left(Mg_{OFM_0} + MNg_{KER_0} + Ng_{IFM_0}\right)} \le P_{in} \le \frac{N\gamma}{Af\left(g_{OFM_0} + Ng_{KER_0} + Ng_{IFM_0}\right)} \tag{12}$$

The lower boundary and upper boundary of P_{in} are V_1 and V_2, respectively. This means that when $P_{in} \in [V_1, V_2] \cap [1, N]$, there must be a $P_{out} \in [1, M]$ that satisfies $MAT = TCT$ and does not exceed the upper boundary of the on-chip buffer. In contrast, if $P_{in} \in (-\infty, V_1) \cap [1, N]$, then there is always $MAT < TCT$, if $P_{in} \in (V_2, +\infty) \cap [1, N]$, then there is always $MAT > TCT$. The first step of the algorithm is using V_1 and V_2 to divide the original search space into three parts. According to the above analysis, we can find the solution that achieves the maximum throughput. The second step is adjusting the solution space. The algorithm calculates the maximum throughputs of each interval and chooses the maximum as the global maximum value.

1) TCT-DOMINATED RANGE

When $P_{in} \in (-\infty, V_1) \cap [1, N]$, $MAT < TCT$, then the throughput of the system is decided by TCT, to find the maximum $P_{in} \times P_{out}$, take the equation case of the formulation 10-3: $\mu_I + \mu_K + \mu_O = Capacity$, and let P_{in} equal P_{out}, and $y(P_{in}) = P_{in}P_{out}$. Formulation (13) obtains the derivative of $y(\cdot)$ about P_{in},

$$y'(P_{in}) = \frac{-g_{IFM_0}K^2P_{in}^2 - 2g_{OFM_0}g_{IFM_0}P_{in} + g_{OFM_0}Capacity}{\left(g_{KER_0}P_{in} + g_{OFM_0}\right)^2} \tag{13}$$

Let $y'(T_n) = 0$, and the right pole of the unary quadratic function can be solved as the maximum value point of the original function T_{n0} (14), as shown at the bottom of the next page.

Whether the maximum value of P_{in} falls in the range $(-\infty, V_1]\cap[1, N)$ and ensures that there is an integer solution for P_{out} satisfying Equation (10-3), we discuss the following: If $P_{in0} \in (-\infty, V_1] \cap [1, N]$, then compare $y(P_{in})|_{P_{in}=1}$ and $y(P_{in})|_{P_{in}=P_{in0}}$, and choose the largest result, If $P_{in0} \in (N, +\infty)$, then compare $y(P_{in})|_{P_{in}=1}$ and $y(P_{in})|_{P_{in}=min(N,V_1)}$, and choose the largest one as the final result, If $P_{in0} \in (-\infty, 1)$, $P_{in} = 1$ is the final result because when $P_{in} > P_{in0}$, $y(T_n)$ decreases monotonically in the interval [1,N].

2) MAT DOMINANT INTERVAL

When $P_{in} \in (V_2, +\infty) \cap [1, N]$, $MAT > TCT$, then the throughput of the system is decided by TCT, and the formula for calculating the MAT cost is shown in Equation (15).

$$MAT = A\left(\frac{1}{T_m}Ng_{IFM_0} + MNg_{KER_0} + Mg_{OFM_0}\right) \tag{15}$$

It can be found that MAT is only related to P_{out}. The larger P_{out} is, the smaller MAT is. Therefore, it is only necessary to keep P_{out} as large as possible. Therefore, let P_{out} =M, and use Equation (10-3) to solve the value of P_{out}, which is the proper partition to maximum throughput.

3) RANDOM DOMINANT INTERVAL

Random dominance occurs when $P_{in} \in [V_1, V_2] \cap [1, N]$, there will be a P_{out} that makes MAT=TCT, and the enumeration method is used to traverse the partition value range and find the partition value.

The above optimal solution is a theoretical result, but in practical systems, due to the storage capacity limitation, some theoretical values may not be reached, which requires adjusting the results to meet system resource constraints. The adjustment scheme of the solution is based on the fact that for any convolutional layer, when $P_{in} = P_{out} = 1$, the storage overhead cannot exceed the memory capacity.

Based on the above analysis, the inverse solution function $P_xV(\cdot)$ is used for the problem that the current partition space has no solution. According to the input partition size, the partition size of another dimension is obtained when the equal sign is taken according to Formula (11-3). For example, the $P_xV(P_n)$ function returns the maximum value of P_{out}, and inputting P_{out} returns the maximum value of P_{in}.

TABLE 4. Matrix reuse distance.

	LFX	LFY	LSX	LSY	LIF	LOF
IFM	$1/K$	$K/1$	$1/K$	$K/1$	1	M
KER	1	1	C	R	1	1
OFM	K	K	1	1	N	1

Taking the solution adjustment of the MAT-dominated interval as an example, the throughput is only related to the value of P_{out}, and P_{out} needs to be as large as possible. However, since P_{in} has a lower boundary and P_{out} cannot achieve the upper boundary of M, we can set $P_{in} = 1$ and use the inverse solution function to obtain the maximum value of P_{out} to reduce traversal trials. If the on-chip buffer is large enough, P_{out} can be set to N to ensure maximum resource utilization.

C. LOOP SCHEDULING SCHEME FOR A SPARSE MATRIX

The data partition is determined before running, and proper scheduling is required for dynamic execution. Eighty percent of the neural network computations are on the convolution layer and fully-connected layer, which are multilayer loops. Therefore, the loop execution order is the key technique for minimizing the data copy operation and the memory access overhead. We propose a memory access footprint model for sparse partitioning and design a traffic traversal algorithm based on the proposed footprint model to find the optimal loop scheduling order.

$$F_X(L_i) = F_X(L_{i-1}) \times \frac{n(L_i)}{R_X(L_i)} \quad (16)$$

Without considering sparseness, the model of the impact of the number of iterations and reuse distance on the memory traffic is as follows. Let the sequence $L_0, L_1, \cdots, L_{N-1}$ be the order of the loop from the inner layer to the outer layer, and the nonrepetitive sampling of $L_i \in \{LSX, LSY, \cdots, LOF\}$, denote $F_X(L_i)$ as the footprint of matrix X in cycle L_i, specifically $F_X(L_{-1}) = 1$, $n(L_i)$ is the number of iterations in the loop L_i, and $R_X(L_i)$ is the reuse distance of the matrix X in the loop L_i. The reuse distance is shown in Table 4, and the footprint calculation formula is shown in Equation (16).

After the matrix is sparsely compressed, denote K_X as the sparse storage footprint factor of matrix X, and the footprint calculation formula is shown in Equation (17)-(18).

$$F_X(L_i) = MIN(F_X(L_{i-1}) \times \frac{n(L_i)}{R_X(L_i)} \quad (17)$$

$$F_X(L_{i-1}) \times \frac{n(L_i)}{R_X(L_i)} \times K_X) \quad (18)$$

K_X is a variable related to the matrix size and sparse algorithm. The sparse matrix footprint factor $\frac{n(L_i)}{R_X(L_i)}$ and the

Algorithm 1 Loop Scheduling Algorithm

Input: dx is matrix average density, $l[K]$ is the set of all scheduling scheme for a loop, px is reuse distance, a is sparse coefficient, b is spare offset, s is the loop order of scheduling scheme of li, which is the i-th item in l, Re is the current footprint recursion factor, pai is multiply recursion factor
Output: min memory consumption $Tmin$ and the corresponding loop sequence

```
/*travers all the loop labels in the current order */
While($li$< K){
  mem=0
  n-loop-step = $l$ [$li$][0]
  While(n-loop-step < CNN-loop-threshold){
    /*Add the matrices of different labels to the data in s according to the corresponding positions */
      $mem+ = Re^*pai$}
      $mem* = \min(mem, mem^*dx^*a + b)$
  }
    s[$li$].traffic= mem
  }
   loop-order = 0
   Tmin = s[0].traffice
   While(loop-order < $K$){
     if(s[loop-order].traffic < Tmin)
                 $Tmin = s[loop - order].traffic$
     }
     Output Tmin and loop-order
```

loop recursion factor $\prod_{j=i}^{N-1} n\left(L_j\right)$ will change with different loop execution sequences and different data partitions. When the matrix loop nesting order changes, the footprint model also changes accordingly. Different matrices have different reuse effects under different loop scheduling. Therefore, it is necessary to calculate the matrix memory footprint for different loop scheduling schemes, as shown in Equations (19) and (20), where $Trace_X$ is the memory footprint of matrix X, Trace is the total memory consumption of memory mapping, and $n\left(L_j\right)$ is the iteration number of loop L_j.

$$Trace_X = F_X(L_i) \times \prod_{j=i}^{N-1} n\left(L_j\right) \quad (19)$$

$$T_{n0} = \left\lfloor \frac{-2g_{OFM_0}g_{IFM_0} + \sqrt{\left(2g_{OFM_0}g_{IFM_0}\right)^2 + 4g_{KER_0}g_{OFM_0}g_{IFM_0}Capacity}}{2g_{IFM_0}g_{KER_0}} \right\rfloor \quad (14)$$

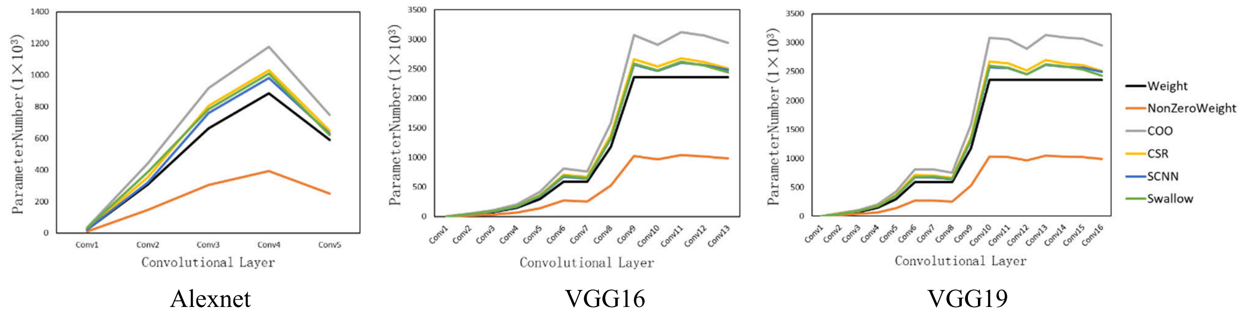

FIGURE 2. Parameter size of kernels under different compression alogrithm.

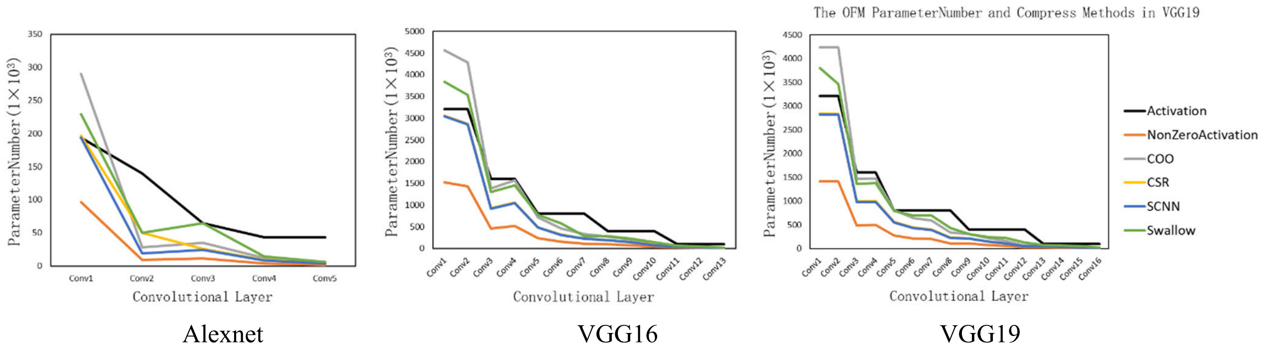

FIGURE 3. Parameter size of OFM under different compression alogrithm.

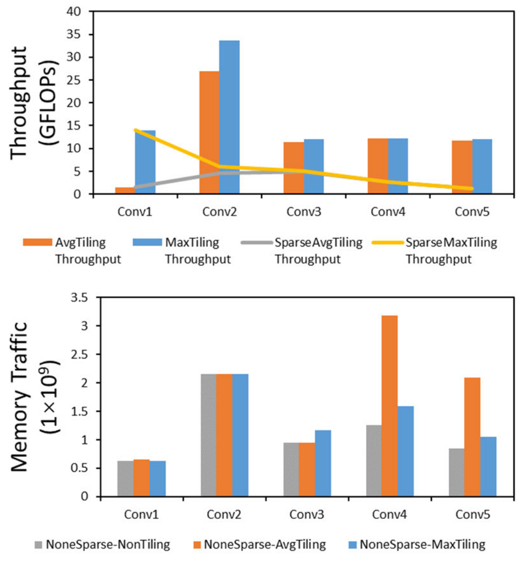

FIGURE 4. Performance of Partition Algorithms on AlexNet.

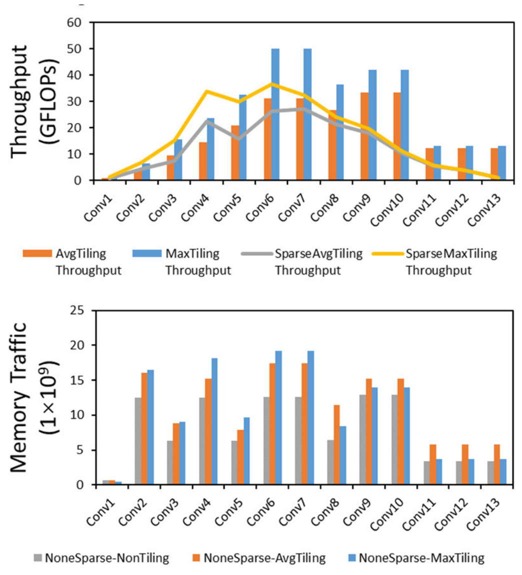

FIGURE 5. Performance of partition algorithms on VGG16.

$$Trace = Trace_{IFM} + Trace_{KER} + Trace_{OFM} \quad (20)$$

The loop scheduling scheme is based on memory consumption, as shown in Algorithm 1.

IV. EXPERIMENTS

We adopt typical convolution networks: AlexNet, VGG and VGG19, and run the ImageNet dataset. First, we evaluate the storage capacity overhead of network parameters under different compression schemes, and the correctness of the sparse feature analysis model is verified. Then, we evaluate the system throughput and memory access under different partition schemes, and the effect of the partition selection algorithm is analyzed. Finally, the memory access of the partition and scheduling joint scheme is tested, and the comprehensive effect of our method is given.

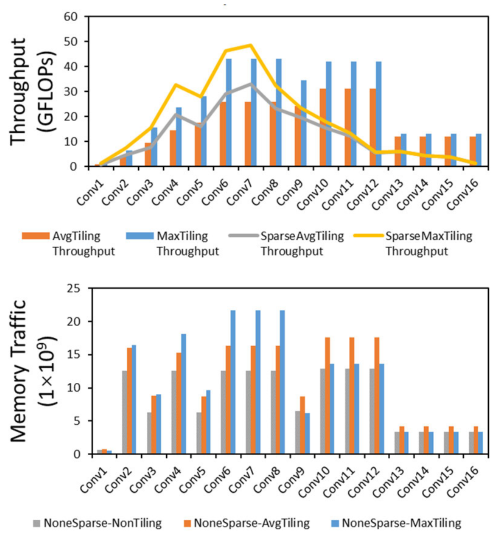

FIGURE 6. Performance of partition algorithms on VGG19.

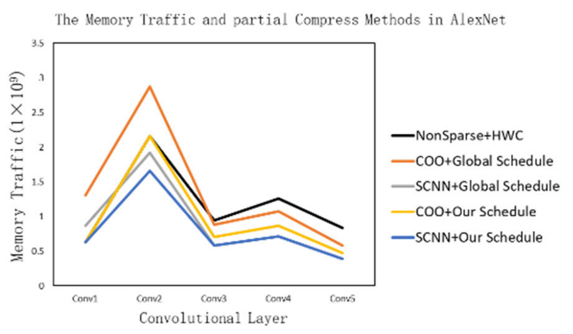

FIGURE 7. Memory traffic of VGG16.

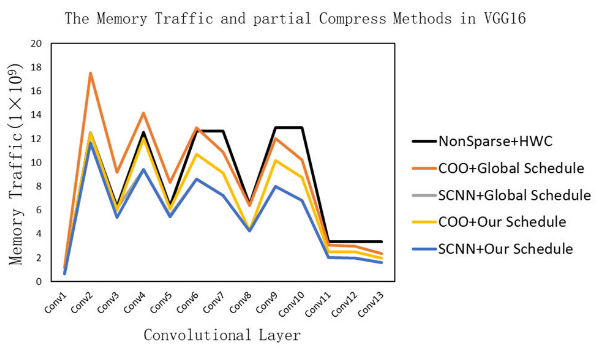

FIGURE 8. Memory traffic of VGG19.

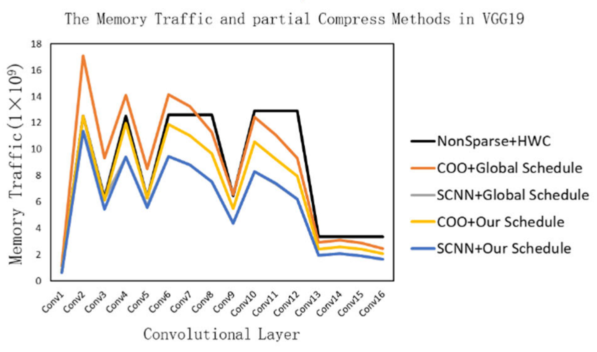

FIGURE 9. Memory traffic of AlexNet.

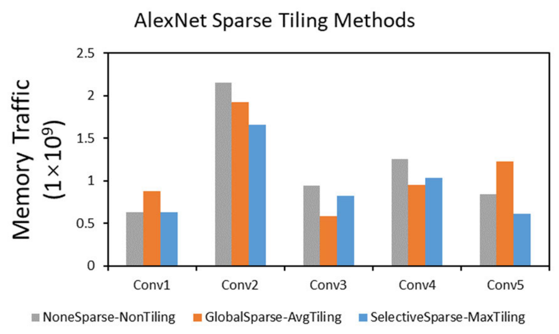

FIGURE 10. Memory Traffic of AlexNet with the joint method.

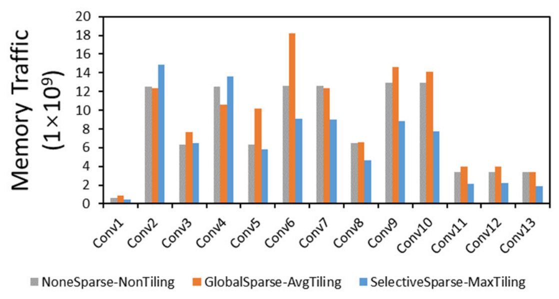

FIGURE 11. Memory traffic of VGG16 with the joint method.

A. MEMORY ACCESS FEATURE ANALYSIS

The memory access features are extracted for the convolution of different layers. Figures2-3 show the memory consumption of four different sparse matrix compression methods for AlexNet, VGG16 and VGG19 networks under the ImageNet dataset. The results show that the sparse matrix compression method can greatly improve memory efficiency.

The improvement of COO is the smallest because each nonzero datapoint needs to be stored with two additional position parameters, which is effective only for matrices whose sparsity exceeds a certain range. SCNN is the most efficient, it generates a serial number for each nonzero data to indicate the number of columns and only stores the number of rows of an additional matrix to indicate the specific location of the data. CSR and Swallow are more balanced than the former two. Although CSR and Swallow cannot achieve the compression effect of the SCNN, there is no restoration problem.

B. PARTITION ALGORITHM EVALUATION

We compared our scheme with the average partition scheme. The average partition scheme does not consider the importance of matrix channels but only considers the matching of the number of input matrices and convolution kernels, so let Tm=Tn to reduce the computational time overhead and increase throughput. This section compares the throughput and memory access overhead of different partition methods at various layers of the network. The results are shown in Figures 4-6.

The following conclusions can be drawn from the experimental results. First, our maximum partition method achieves higher system throughput than the average partition method. Especially in the hidden layer of the neural network, the maximum partition method can obtain the partition result corresponding to the optimal throughput of the current network layer, and the memory access cost and computation cost are low, while the average partition method can only reduce the computation cost. The memory access overhead is still high.

Second, the throughput-based maximum partition algorithm proposed in this paper is better optimized for deep convolutional neural networks. Since a deeper network has a larger scheduling space, the layers close to the output have a more obvious effect on the improvement of throughput and the reduction in memory access.

C. DATA PARTITION AND LOOP SCHEDULING JOINT EVALUATION

We compared the effectiveness of the scheduling scheme with HWC, and the results are shown in Figures7-9. As seen in the figures, the method in this paper further reduces the overall memory access of the convolutional neural network. However, because the layer close to the input side has a high density, the advantages of the sparse matrix compression algorithm cannot be exerted, so the optimization result is not obvious, and the layer close to the output has a low matrix density due to more 0 values. At this time, the optimization effect of the sparse compression algorithm is obvious. Experiments show that, compared with the HWC scheme, using the compression algorithm with a sparse coefficient of 3, the average memory access of each layer is compressed to 83% in the AlexNet, VGG16 and VGG19 networks, while using the compression algorithm with a sparse coefficient of two, memory access is compressed to 68% of the original.

Figures 10-11 show the joint effect of the maximum partition scheme based on the sparse matrix and the loop scheduling scheme. The results show that in the AlexNet network, the throughput is increased by 1.95 times, the throughput of VGG is increased 1.38 times on average, the overall throughput is increased by 1.66x on average. The memory access is reduced by 14.6% on average, which verifies the effectiveness of our scheme.

V. CONCLUSION

With the development of neural networks, the complexity of the network model and the model parameters are increasing, which makes it difficult to save all the data on the limited memory of embedded or mobile devices. Although previous works achieved certain acceleration optimization effects, they ignored the impact of matrix sparsity on loop scheduling. How to use the matrix sparse compression algorithm has become the primary obstacle to solving the problem. We proposed a sparse neural network data partition and loop scheduling scheme. We extracted the sparse feature of the matrix and analyze the characteristics of the sparse compression algorithm. Based on the sparsity feature of the matrix, we selected the proper partition size to maximize throughput. Then, we used the loop sequential traversal algorithm to find the proper loop scheduling scheme using the chosen partition size. The experimental results show that the average memory access of each layer can be compressed to 68% of the original, additionally, the throughput of the three networks increases to an average of 1.66 times

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–9.

[2] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proc. Conf. High Perform. Comput. Netw., Storage Anal.*, 2009, pp. 1–11.

[3] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, Jun. 2017, pp. 27–40.

[4] B. Liu, X. Chen, Y. Han, and H. Xu, "Swallow: A versatile accelerator for sparse neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4881–4893, Dec. 2020.

[5] C. Y. Hitchcock and D. E. Thomas, "A method of automatic data path synthesis," in *Proc. 20th Design Autom. Conf.*, 1983, pp. 484–489.

[6] A. C. Parker, J. Pizarro, and M. Mlinar, "MAHA: A program for datapath synthesis," in *Proc. 23rd ACM/IEEE Design Autom. Conf.*, Jun. 1986, pp. 461–466.

[7] B. M. Pangrle and D. D. Gajski, "Slicer: A state synthesizer for intelligent silicon compilation," in *Proc. Int. Conf. Comput.-Aided Design*, Oct. 1987, pp. 42–45.

[8] Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. C-30, no. 7, pp. 478–490, Jul. 1981.

[9] S. Davidson, D. Landskov, B. D. Shriver, and P. W. Mallett, "Some experiments in local microcode compaction for horizontal machines," *IEEE Trans. Comput.*, vol. C-30, no. 7, pp. 460–477, Jul. 1981.

[10] R. Camposano, "Path-based scheduling for synthesis," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 10, no. 1, pp. 85–93, Jan. 1991.

[11] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 6, pp. 661–679, Jun. 1989.

[12] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, Jun. 2017, pp. 535–547.

[13] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using Halide's scheduling language to analyze DNN accelerators," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2020, pp. 369–383.

[14] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[15] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2015, pp. 161–170.

[16] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–9.

[17] R. Fang, J. Wang, Z. Yao, C. Liu, and W. Zhang, "Modeling computational feature of multi-layer neural network," *J. Comput. Res. Developmen*, vol. 56, no. 6, pp. 1170–1181, 2019.

[18] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for FPGA-based convolutional neural networks," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–8.

[19] A. Stoutchinin, F. Conti, and L. Benini, "Optimally scheduling CNN convolutions for efficient memory access," 2019, *arXiv:1902.01492*.

[20] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 343–348.

[21] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
[22] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-optimized FPGA accelerator for deep convolutional neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 3, pp. 1–23, Jul. 2017.
[23] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 25–34.
[24] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 553–564.
[25] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2010, pp. 247–257.
[26] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 32–37.
[27] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *Proc. CVPR Workshops*, Jun. 2011, pp. 109–116.
[28] 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," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Feb. 2014.
[29] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.
[30] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 92–104.
[31] S. I. Venieris and C.-S. Bouganis, "FpgaConvNet: A framework for mapping convolutional neural networks on FPGAs," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2016, pp. 40–47.
[32] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *ACM Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, vol. 53, pp. 461–475, Mar. 2018.
[33] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "AutoDNNchip: An automated DNN chip predictor and builder for both FPGAs and ASICs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2020, pp. 40–50.
[34] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "VDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.
[35] S. Li, X. Shen, Y. Dou, S. Ni, J. Xu, K. Yang, Q. Wang, and X. Niu, "A novel memory-scheduling strategy for large convolutional neural network on memory-limited devices," *Comput. Intell. Neurosci.*, vol. 2019, pp. 1–12, Apr. 2019.
[36] R. V. W. Putra, M. A. Hanif, and M. Shafique, "DRMap: A generic DRAM data mapping policy for energy-efficient processing of convolutional neural networks," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[37] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

DEJIAN LI received the master's degree in electronics from Tsinghua University, in 2002. He is currently working at Beijing Smartchip Microelectronics Technology Company Ltd. He focuses on very large scale ASIC design and verification, especially in the fields of industrial control. He has published ten technical papers and ten patents in related areas.

RONGQIANG FANG received the master's degree in electronics from Capital Normal University, in 2021. He is currently pursuing the Ph.D. degree in computer application technology with Beijing Jiaotong University, China. His research interests include computer architecture and fault-tolerant design.

JING WANG received the Ph.D. degree from Peking University, in 2011. She is currently an Associate Professor with the School of Information, Renmin University of China. Her research interests include computer architecture, energy-efficient computing, high-performance computing, and hardware reliability and variability.

DONGYAN ZHAO received the master's degree from Shanghai Jiao Tong University, in 1998. She is currently an Executive Director of Beijing Smartchip Microelectronics Technology Company Ltd. Her research interest includes integrated circuit design. She is also a National Candidate of "Millions of Talents Project." She has published 54 patents, 77 papers, and six books.

TING CHONG received the master's degree from Tianjin University, in 2007. He is currently the Technical Expert of Beijing Smartchip Microelectronics Technology Company Ltd. His research interests include embedded CPU technology, embedded software and hardware technology, information security technology, and functional security technology.

ZENGMIN REN received the master's degree from Hangzhou Dianzi University, Hangzhou, China, in 2008. He is currently working at Beijing Smartchip Microelectronics Technology Company Ltd. His research interests include embedded CPU technology, information security technology, and functional security technology.

JUN MA received the master's degree in computer application from Inner Mongolia University, in 2007. He is currently an Engineer at Beijing Smartchip Microelectronics Technology Company Ltd. His research interests include hardware secure architecture and trusted execution environment design in embedded field.