DASC : A DRAM Data Mapping Methodology for Sparse Convolutional Neural Networks

Bo-Cheng Lai, Tzu-Chieh Chiang, Po-Shen Kuo, Wan-Ching Wang, Yan-Lin Hung, Hung-Ming Chen, Chien-Nan Liu and Shyh-Jye Jou
Institute of Electronics
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
bclai@nycu.edu.tw, m56565566.ee08@nycu.edu.tw, kuoposhen@gmail.com, rockruby622@gmail.com, stevenhungs@gmail.com, hmchen@mail.nctu.edu.tw, jimmyliu@nctu.edu.tw, jerryjou@mail.nctu.edu.tw

***Abstract*—The data transferring of sheer model size of CNN (Convolution Neural Network) has become one of the main performance challenges in modern intelligent systems. Although pruning can trim down substantial amount of non-effective neurons, the excessive DRAM accesses of the non-zero data in a sparse network still dominate the overall system performance. Proper data mapping can enable efficient DRAM accesses for a CNN. However, previous DRAM mapping methods focus on dense CNN and become less effective when handling the compressed format and irregular accesses of sparse CNN. The extensive design space search for mapping parameters also results in a time-consuming process. This paper proposes DASC: a DRAM data mapping methodology for sparse CNNs. DASC is designed to handle the data access patterns and block schedule of sparse CNN to attain good spatial locality and efficient DRAM accesses. The bank-group feature in modern DDR is further exploited to enhance processing parallelism. DASC also introduces an analytical model to facilitate fast exploration and quick convergence of parameter search in minutes instead of days from previous work. When compared with the state-of-the-art, DASC decreases the total DRAM latencies and attains an average of 17.1x, 14.3x, and 23.3x better DRAM performance for sparse AlexNet, VGG-16, and ResNet-50 respectively.**

Keywords—sparse CNN, DRAM, data mapping, optimization, design space exploration

I. Introduction

Convolution Neural Networks (CNN) has shown promising results in various intelligent applications [1, 2]. The sheer data volume of network models and high arithmetic intensity have made CNN processing both computation and data intensive. Customized accelerators alleviate the computation bottleneck [3, 4], but the massive data communication between accelerators and DRAM still cost substantial processing time and energy consumption [3, 5].

Pruning [6] was proposed to trim down the non-effective neurons and significantly reduce the CNN model sizes and the resulting DRAM accesses. Although the irregular data access patterns of sparse networks can be managed by the accelerators [7], the DRAM accesses of the non-zero data in a sparse network still dominate the overall system performance. It has been shown that over 80% of the system runtime and energy consumption are caused by DRAM accesses [3, 4, 8].

Data mapping methodologies were used to facilitate efficient DRAM data accesses and minimize latency and energy consumption [9, 10]. For example, DRMap [9] maps dense CNN models to DRAM with appropriate layer partitioning and adaptive reuse priority, and shows 96% reduction of DRAM accesses energy delay product (EDP). However, there still exist some pending challenges for the current DRAM data mapping approaches of CNNs. First, the characteristics of sparse CNN networks are not taken into account. The existing DRAM mapping methods show less effective DRAM accesses for compressed formats of sparse CNNs. Second, existing approaches [9] focus mainly on DDR3 protocol and do not exploit the newly introduced DRAM architectures, such as bank-groups [11], in DDR4 or DDR5. Third, previous mapping approaches rely on extensive search for mapping parameters. The time-consuming design space exploration not only causes long turn-around time but also poses limitation on the thorough evaluation of parameters.

In this paper, we propose DASC, a DRAM data mapping methodology for both sparse and dense CNNs. DASC enables efficient accesses within a data tile in DRAM and schedules the access sequence of data blocks to attain better spatial locality and row buffer hit rates. The bank-group features of modern DDR architecture are exploited to attain higher access parallelism and bandwidth. To facilitate fast exploration of enormous mapping parameters, an analytical model is introduced to support to quickly converge the search of data tile parameters from over a day to tens of seconds. The experimental results have shown that DASC attains an average of 17.1x, 14.3x, and 23.3x better DRAM performance for sparse AlexNet, VGG-16, and ResNet-50 respectively, when compared to the state-of-the-art mapping scheme DRMap [9].

The rest of the paper is organized as follows. The background of CNN processing, sparse networks, DRAM organization, and previous works of data mapping will be introduced in Section II. The proposed DASC mapping for sparse CNN will be introduced in Section III. Section IV elaborates the experimental setup and the results will be discussed in Section V. Section VI will the conclusion.

II. Background and Related Works

A. Sparse CNN and Compressed Format

Fig. 1(a) shows an example of the convolution layer. A CNN layer contains a weight filter map of size M x M with C channels and depth D. With an input activation map of size H x W with C channels, the convolution operation is performed between the input activation map and the weight filter map shown as Fig. 1(a). These operations will generate an element in the output activation map of size K x L with D channels. A sliding window will pass through the input activation map and generate all the output activation maps.

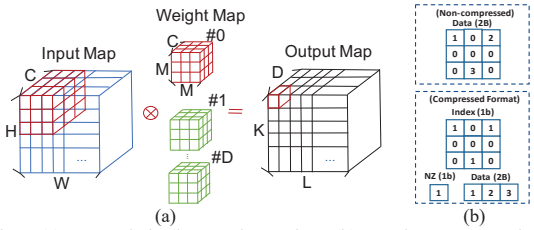

Fig. 1. (a) A convolution layer and operations. (b) Encode a sparse matrix (top) into the compressed data format (bottom). Note that both Index and NZ are type of one bit, while data is of 2Byte.

Sparse CNN is generated by the model refinement techniques, such as pruning [6], to trim down less effective weights and reduce model size. The sparse CNN is stored in a compressed data format, which keeps Index, bit vectors of Non-Zero blocks (NZ), and non-zero data [8], as shown in Fig. 1(b). Index uses one bit to mark each of the non-zero element. NZ uses one bit to mark if a data block contains any non-zero elements and can be used to skip the data blocks with all zeros. With the encoded format, the required storage space can be significantly reduced if the network sparsity is high. However, previous studies [3, 5, 8] still show that the DRAM accesses dominate the performance even with a high sparsity CNN model encoded in the compressed format.

B. DRAM Organization and Operations

Fig. 2(a) shows the structure of the modern DRAM, which can be hierarchically organized into channel, rank, chip, bank-group, bank, row and column. A rank consists of several DRAM chips where each DRAM chip contains multiple bank groups. Each bank (in Fig. 2(b)) in a bank group is made up of a row decoder, a column decoder, a row buffer and a cell-array. Note that the hierarchy of bank-group [12] exists in modern DDR standards, such as DDR4 and DDR5. Each bank-group can be independently operated and increase internal throughput by supporting parallel data accesses.

When there is a DRAM access request, the decoder will guide it to the corresponding bank and split the address into the row address and column address. The ACTIVE command will first activate the target row and the data will be loaded to the row buffer. Then a READ/WRITE command with column address can be issued to the accessible data in the row buffer. Note that the data of an already activated row in the row buffer can be accessed without the need to send an additional ACTIVE command. This situation is called a *row buffer hit*. If the row buffer is empty or stores the data of another row, it is referred as *row buffer miss* or *conflic*t. In these cases, it will take long latency to access the target data since the current opened row needs to be closed first before sending the ACTIVE command to activate the target row.

C. Previous Works of Data Mapping for DRAMs

The performance of DRAM accesses can be enhanced if the accessed data are properly mapped. Previous data mapping techniques can be divided into hardware and software approaches. Hardware approaches [13, 14] aim to modify the DRAM architecture, and provide extra hardware like middle buffer, additional controller, and isolation transistors to improve energy-efficiency and parallelism. However, the fixed design of hardware implementation limits the support for specific applications only.

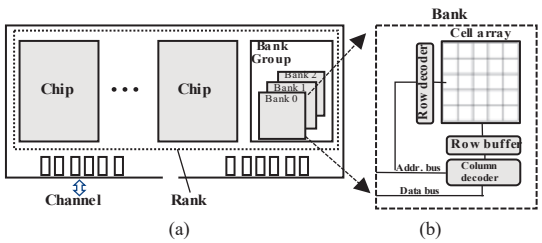

Fig. 2. DRAM system organization. (a) A DIMM (Dual In-line Memory Module) contains several DRAM chips. Each DRAM chip contains multiple bank-groups. (b) A bank in a bank-group contains a row decoder, a column decoder, a row buffer and a cell-array.

Recent works of software data mapping approaches [9, 15] conduct adaptive partitioning methodology to achieve the minimal DRAM access volume with the support of limited on-chip memory. DRMap [9] is a software approach on DRAM mapping for CNN models. DRMap introduces a tile-based mapping scheme to increase DRAM row buffer locality and attain low EDP (Energy-Delay-Product). However, DRMap mainly focuses on dense CNN models and cannot properly map the data of sparse CNNs. The accesses to the compressed format could generate multiple requests to distinct DRAM addresses and cause high access overhead. The coarse-grained data layout of the one-bit data type of Index and NZ (in Fig. 1(b)) could also return low amount of effective data for each DRAM transfer. The features of modern DDR standards, such as bank-group parallelism, is not exploited in DRMap either. The extensive parameter search of DRMap also causes time-consuming parameter searching.

III. DASC: A DRAM Data Mapping Methodology for Sparse Convolutional Neural Networks

A. Overall Flow of DASC Methodology

In this paper, we propose DASC, a DRAM data mapping methodology for sparse CNN. DASC takes a top-down approach to first analyze the tiling factors of sparse CNN, then find data block scheduling, and finally determine detail data mapping. As shown in Fig. 3, the DASC flow can be divided into three stages. In Stage 1, we introduce an analytical model to quickly estimate the impact of tiling factors and return a tiling scheme which can fit data blocks to on-chip buffers. In Stage 2, we choose a proper block schedule which can attain high data locality. In Stage 3, we map the data to DRAM according to the schedule from Stage 2, and exploit both bank-group and bank parallelism in modern DDR structure (e.g. DDR4) to improve the throughput by proper data mapping for sparse CNN. The detail of each stage will be elaborated in the following sections.

B. Stage 1: A Fast Method to Find Tiling Factors

When loading the CNN data from DRAM to on-chip buffers of limited capacity, a general design guideline is to tile the data and perform the computation on the data tiles [9]. Enhancing data reusability of on-chip buffers would reduce the DRAM data accesses. The method in [9] searches all the possible combinations of tiling factors, and could take more than a day to find the proper tiling factors among 3.2 million parameter combinations of VGG16. In this paper, we propose an analytical model (in Algorithm 1) to quickly converge the design space exploration and find proper tiling factors that form the data blocks to fit in the on-chip buffer.

In Algorithm 1, the analysis will be performed for Index, Data, and NZ buffer. The inputs contain four types of information, including CNN convolution layer configuration, data size, block length, and on-chip buffer size. The Index is analyzed from Line 2 to Line 9. We first calculate the

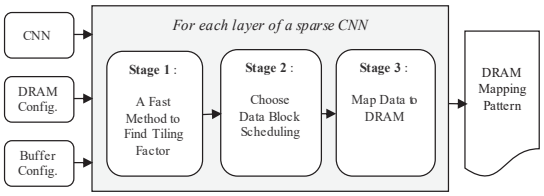

Fig. 3. Design flow of DASC methodology.

maximum number of weight index that can fit in weight index buffer, then use the result to find the index channel tiling factor Tic by dividing the weight window (Line 4). After that, we find the maximum number of input index that can fit in input index buffer and find the number of window index in Line 5-6. Line 7 constraints the element size not exceeding the buffer size and Line 8 finds the maximum Tih and Tiw.

Algorithm 1: FIND TILE PARTITION FACTORS

INPUT: CNN Configuration: Input Width (W), Input Height (H), Input Channel (C), Input Stripe (S), Weight Length (M), Weight Depth (D), Output Width (K), Output Height (L);
Buffer Sizes (BSize), Data Sizes and Block Length (block_L)
OUTPUT: Tiling Factors of Channel (Tc), Width (Tw), Height (Th);
1: **Begin:**
2: **Analyzing Index:**
3: NumWIndex ← WIBSize / IS; // NumWIndex: number of weight index
4: Tic ← NumWIndex / (M*M*C); // M:Weight Length, C:Input Channel
5: NumIIndex ← IIBSize / IS; // NumIIndex: number of input index
6: WinIndex ← (M + S * Tih) ^ 2; //WinIndex: number of window index
7: NumIIndex >= WinIndex * C // elements fit in buffer
8: solve max Tih < (sqrt (NumIIndex / C) – M) / S;
9: Tiw ← Tih;
10: **Analyzing Data:**
11: NumWData ← WDBSize / DS; // NumWData: number of weight data
12: Tdc ← NumWData / (M*M*C);
13: NumIData ← IDBSize / DS; // NumIData: number of input data
14: WinData ← (M + S * Tdh) ^ 2; //WinIndex: number of window data
15: NumIData >= WinData * C; // elements fit in buffer
16: solve max Tdh < (sqrt (NumIData / C) – M) / S;
17: Tdw ← Tdh;
18: **Analyzing NZ:**
19: NumWNZ ← WNBSize / NS; // NumWNZ: number of weight NZ
20: Tnc ← NumWNZ / (M*M*C*block_L*block_L);
21: NumINZ ← INBSize / NS; // NumINZ: number of input NZ
22: WinNZ ← ((M+S*Tih)/block_L)^2; //WinNZ : number of window NZ
23: NumINZ >= WinNZ * C; // elements fit in buffer
24: solve max Tnh < (sqrt (NumINZ / C) – M + block_L) / S;
25: Tnw ← Tnh;
26: **Finding Tiling Factor:**
27: Tc ← min(Tic, Tdc, Tnc);
28: Tw ← min(Tiw, Tdw, Tnw);
29: Th ← min(Tih, Tdh, Tnh);
30: **Return** Tc, Tw, Th;
31: **End**

The analysis process of Data (Line 10-17) is similar to the process of Index. The main difference is that the data size DS is 2 bytes instead of 1bit as in the Index analysis. The process of NZ (Line 18-25) is also similar and considers the block length (block_L) during the analysis. By searching (Line 27-29) the generated Tc, Tw, and Th, the algorithm will output the maximum Tc, Tw, Th that can fit in on-chip buffers. These parameters will then be forwarded to the next stage of DASC to determine the block scheduling (Stage 2) and detail data mapping (Stage 3).

C. Stage 2: Choose A Proper Data Block Scheduling

The compressed data format contains Index, NZ bit vector, and non-zero data. These components are packed into the unit of a *data block*, and processed together. The execution schedule of data blocks determines how well the data on the buffer can be reused between the processing of data blocks.

To simplify the complexity of scheduling, DASC adopts a data block scheduling scheme which assembles all the possible schedules based on dimensions of each data type and only reserves the data schedule offering maximum spatial locality in our analytical estimation. The scheduling algorithm is described in Algorithm 2. For each layer of the network (Line 1), all the possible priority orders of the input data directions will be generated and saved as the options for DRAM data mapping. The configuration (d^n, p^n) of the current layer is first collected (Line 2) where d^n denotes the information of each direction and p^n represents the number of data blocks in each direction after partitioning. Then the block configuration p^n maps the value to corresponding range factor (p_{1st}^n, p_{2nd}^n, p_{3rd}^n) according to scheduled directions order ds^n in Line 3-5. Based on the known direction priority and range of data block, the block index ps^n are produced in the specified order in Line 6. By accumulating consecutive block index in Line 6-7, all possible data block orders $Q^n(ds^n)$ are formulated.

Algorithm 2: DATA BLOCK SCHEDULING

1: **for** each layer n ∈ N **do**
2: configuration_collector (d^n, p^n)
3: **for** each sequence $ds^n \in d^n$ **do**
4: $Q^n(ds^n)$ ← data block order
5: (p_{1st}^n, p_{2nd}^n, p_{3rd}^n) ← (ds^n, p^n)
6: **for** each state $ps^n = (1{:}p_{1st}^n, 1{:}p_{2nd}^n, 1{:}p_{3rd}^n)$ **do**
7: $Q^n(ds^n) \leftarrow Q^n(ds^n) + ps^n$
8: **end for**
9: **end for**
10: **end for**

Note that this scheduling scheme needs to be performed on all the CNN model components, including input feature maps, weight, and output feature maps. For input feature map, the data is represented with 3-way tensors that contains the directions on depth, width, and height. These directions are the main factors that determine the subsequent data block access for the input feature map. For example, in Fig. 4(a), if the priority order of data block mapping is determined as (1) width direction → (2) depth direction → (3) height direction, the algorithm will prioritize the mapping of data blocks in the width direction first. The data block in the depth direction will only be mapped after all the blocks in the width direction have been mapped. When mapping weights, there are four possible directions including height, width, depth, and filter group. The height and width of a kernel are generally small and the tiling sizes of height and width are usually the same as the size of kernels to simplify the complexity of tiling space. Hence, we only need to consider the depth and filter group directions. The algorithm of weight mapping is similar to Algorithm 2, but the weight scheduling only needs to take two different directions into consideration.

There are two different schemes of data block schedules, tile-based (in Fig. 4(b)) and map-based (in Fig. 4(c)). The tile-based scheme schedules the data blocks on the tile direction first. As in Fig. 4(b), the numbers in data blocks denotes the order of the data blocks in the schedule. For example, data block 1 will be scheduled earlier then data block 2. This is a similar adopted in DRMap [9]. Map-based scheme, on the other hand, will schedule the data blocks on the map direction first, as illustrated in Fig. 4(c). These two schemes show different impact when mapping CNN networks. We will discuss the choice of the two schemes in Section III.E.

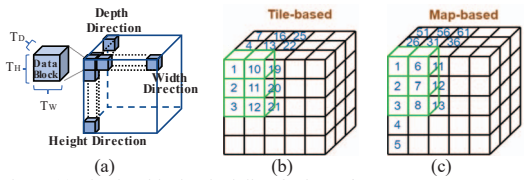

Fig. 4. (a) The data block scheduling for input feature map. T_D, T_H, T_W are block factors (b) tile-based mapping (c) map-based mapping

The output of the algorithm is the result of the convolutional operations between a data block of input data and a data block of weight. The generated block sequence is determined depending on the convolutional processing order of input data and weight. Hence, similar to the input feature map, the output feature map is a 3-way tensor composed of the following directions: depth direction, width direction and height direction. The algorithm of output feature map scheduling is similar to Algorithm 2.

D. Stage 3: Map Data to DRAM

In Stage 3, we retrieve the schedule from Stage 2, and use the schedule to perform data mapping. The schedule from Stage 2 provides the order of data blocks, and will be used for the data mapping. The data mapping will be performed for input feature map, weight map, and output feature map respectively. As in Algorithm 3, the data mapping process will follow the DRAM hierarchy in the following order, from bank-group, column, bank, row, rank, and finally channel. We put the iteration of rows at outer loop to achieve higher row buffer hit rate. The mapping for bank-group is in the inner-most loop to make bank-group switching more frequently and leverage the bank-group parallelism.

Algorithm 3: Process of Data MAPPING

```
INPUT: (1) Data (2) DRAM Configuration
OUTPUT: DRAM Address
1: Begin:
2:   for ch=1 to #ch do              // channel count
3:     for ra=1 to #ra do            // rank count
4:       for ro=1 to #ro do          // row count
5:         for ba=1 to #ba do        // bank count
6:           for co=1 to #co do      // column count
7:             for bg=1 to #bg do    // bank-group count
8:               DRAM[ch, ra, bg, ba, ro, co] ← data;
9:           end for
10:    (…skip layers of end for…)
11: end for
12:Return DRAM Address;
13:End
```

E. Select Direction of Data Mapping

We have observed that, to attain better performance, we need to choose between tile-based and map-based schemes depending on the network property. TABLE I lists the average of the normalized DRAM performance. DRAM performance is calculated as the reciprocal of total DRAM access latencies. DASC$_{tile}$ and DASC$_{map}$ denote DASC scheme with tile-based and map-based mapping respectively. In TABLE I, DASC$_{map}$ returns better performance for networks of large window sizes (e.g. 3x3 in AlexNet), while DASC$_{tile}$ shows better performance for smaller window sizes (e.g. 1x1 in ResNet50). This is mainly caused by the amount of data overlap between convolution windows. Compared to AlexNet/VGG16, ResNet50 has many layers with small weight kernel window size (i.e. 1x1). When stripe is smaller than the kernel width (e.g. AlexNet and VGG16), the map-based mapping could exploit data overlaps and attain higher data reuse than tile-based mapping. For small weight kernels, such as 1x1 in ResNet50, there is no data overlap between convolution windows. In this case, tile-based mapping could benefit from placing the data the same way the accelerator accesses them (i.e. in units of tiles).

TABLE I. AVERAGE OF THE NORMALIZED DRAM PERFORMANCE OF TILE-BASED AND MAP-BASED MAPPING SCHEMES

Schemes	AlexNet	VGG16	Resnet50
DASC$_{tile}$	0.83	0.33	1.54
DASC$_{map}$	1.00	1.00	1.00

TABLE II. CONFIGURATION OF MEMORY FOR CNN ACCELERATOR

Items	Description	
Memory Controller	Page policy	Open-page
	Policy Scheduler	FRFCFS
DRAM	Speed grade	DDR4-2400R
	Organization	DDR4 4Gb x8
	Burst length	8
Buffers	Data buffer	64KB
	Index buffer	4KB
	NZ buffer	4KB

To facilitate the fast and effective design flow, in Stage 2 of DASC, we simply choose DASC$_{map}$ for network layers of window sizes of 3x3 or larger. DASC$_{tile}$ is adopted for layers of 1x1 windows. The more comprehensive study of the impact of different mapping schemes will be in our future work.

IV. EXPERIMENTAL SETUP

We evaluate the performance and energy consumption of the DRAM on Ramulator [16] and DRAMPower [17] respectively. Ramulator is an accurate DRAM simulator which takes the DRAM traces and returns DRAM statistics. DRAMPower receives the command traces from Ramulator and provides DRAM power statistics. The DRAM traces are generated by simulating the data access patterns of CNN accelerators when considering input (CNN model), accelerator configurations (DRAM, buffer size), and data mapping (tiling factors, block schedules, and data mapping).

The system configurations of our experiments are listed in TABLE II. We use DDR4-2400R 4Gb x8 [12] and adopt first-ready first-come first-serve (FRFCFS) as DRAM scheduling policy. This scheduling policy will set high priority to the ready requests that meets all the DRAM timing constraints. The memory controller applies an open-page policy which makes the accesses in the same row faster and more efficient.

We perform experiments on sparse CNN models of AlexNet [18], VGG-16 [19], and ResNet50 [20]. We use Caffe-Python [21] to extract the sparse weights and activations, and input images from ILSVRC12 [22]. The processing is performed on a machine with Intel Xeon W-2125 CPU and 16 GB of DRAM.

In the experiments, we compare the DRAM performance of different mapping schemes. As in TABLE III, DRMap [9] is the state-of-the-art CNN data mapping methodology which adopts the tile-based mapping approach. DRMapS is an enhanced implementation in which we add the support of sparse CNN format to the original DRMap. DASC applies adaptive mapping which chooses a proper method between tile-based and map-based schemes. In addition to the support of sparse CNN format, DASC poses extra benefits of bank-group interleaving to exploit bank-group parallelism. Section V will show the experimental results and comprehensively compare the performance of different designs.

TABLE III. COMPARISON OF DRAM MAPPING SCHEMES

Schemes	Techniques
DRMap [9]	tile-based mapping
DRMapS	tile-based mapping + handling sparse format
DASC (our work)	adaptive mapping + handling sparse format + bank-group parallelism

Note: adaptive mapping will choose a proper mapping between tile-based and map-based scheme.

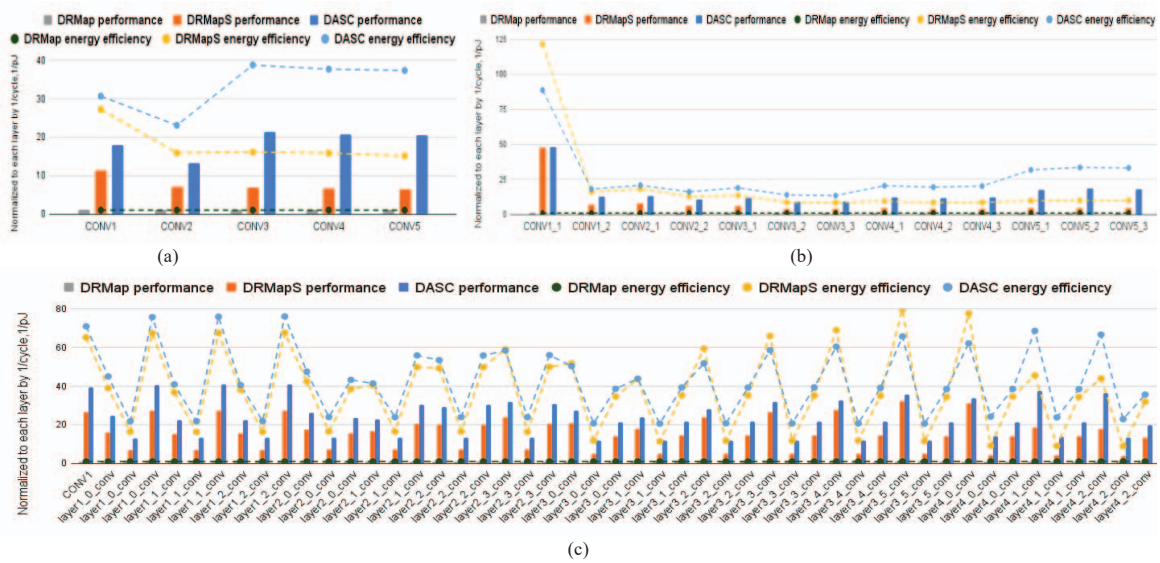

Fig. 5. Normalized DRAM performance and normalized energy efficiency for different mapping schemes. All the values are normalized to DRMap. (a) Sparse AlexNet, (b) Sparse VGG16 (c) Sparse ResNet50

V. Results and Discussion

A. Performance and Energy on DRAM Accesses

We compare the normalized performance of DRAM accesses when processing sparse AlexNet (Fig. 5(a)), sparse VGG16 (Fig. 5(b)), and sparse ResNet50 (Fig. 5(c)). The performance of DRAM is measured as the reciprocal of the total cycles of DRAM accesses. Overall, the proposed DASC achieves from 13.2x to 21.5x higher DRAM performance in AlexNet, 9.19x to 47.9x in VGG16, and 11.9x to 40.9x in RenNet50, when compared with DRMap. When compared with DRMapS which handles sparse CNNs (introduced in Section IV.D), DASC can still achieve up to 3.24x, 4.29x, 3.43x of DRAM performance respectively in AlexNet, VGG16, and ResNet50.

Fig. 5 also shows the energy efficiency of DRAM accesses for different data mapping schemes. The energy efficiency is defined as the reciprocal of energy consumption. DASC can achieve from 23.1x to 38.8x of energy efficiency in AlexNet, 13.4x to 88.8x in VGG16, and 20.6x to 76.2x in RenNet50, when compared with DRMap. When compared with DRMapS, DASC attains up to 2.47x, 3.39x, and 2.57x energy efficiency for AlexNet, VGG16, and ResNet50 respectively.

B. Perforamnce Characteristics of Data Mapping Schemes

In this section, we use various performance indicators to analyze the behavior of different mapping schemes. The superior DRAM performance of DASC is from the high row buffer hit rates of DASC. Recall that the row buffer hit happens when accessing the data of an already activated DRAM row without the need to send additional activation command. The Row Buffer Hit Rate (RBHR) is defined as the ratio of the number of row buffer hits to the total number of row buffer accesses, as listed Equation (1). RBHRs of different data mapping schemes are listed in TABLE IV.

$$\text{RBHR} = \frac{Number\ of\ row\ buffer\ hits}{Number\ of\ total\ row\ buffer\ accesses} \times 100\% \quad (1)$$

In TABLE IV, DASC shows the highest RBHRs among different designs. The RBHR of DRMap are the lowest since it does not apply proper data mapping on sparse CNN and causes non-efficient DRAM data accesses with frequent row switching. Moreover, since the Index and NZ are type of one bit per element, simply allocating these data aligned with the nominal size of a data transaction could return very low amount of useful data, and would cause enormous extra number of data accesses. As shown in TABLE IV, DRMap shows much more DRAM accesses (Access# in TABLE IV) when compared to DASC.

Another indicator for the DRAM performance is the Bank-group Interleaving Ratio (BIR) as in Equation (2). Higher BIR represents higher bank interleaving parallelism, which results in higher DRAM access bandwidth.

$$\text{BIR} = \frac{Number\ of\ bankgroup\ interleaving}{Number\ of\ total\ DRAM\ access} \times 100\% \quad (2)$$

DASC can attain the highest BIR among different schemes. DRMapS shows very low BIR because it did not consider bank-group interleaving and stored continuous data in the same bank-group.

We also compares the DRAM Cycles per DRAM Access (DCDA) among different schemes, to find out the DRAM access efficiency of each algorithm. DCDA is defined in (3).

$$\text{DCDA} = \frac{Number\ of\ total\ DRAM\ cycle}{Number\ of\ total\ DRAM\ access} \quad (3)$$

TABLE IV. DRAM Perforamnce Characteristics of Different Data Mapping Schemes

		DRMap	DRMapS	DASC
RBHR	AlexNet	5.6%	95.0%	97.4%
	VGG16	6.3%	92.7%	92.5%
	ResNet50	5.8%	97.6%	98.1%
Access#	AlexNet	$33x10^6$	5×10^6	3×10^6
	VGG16	$560x10^6$	135×10^6	74×10^6
	ResNet50	$167x10^6$	204×10^6	14×10^6
BIR	AlexNet	70.7%	1×10^{-4}%	98.3%
	VGG16	69.9%	1×10^{-4}%	94.0%
	ResNet50	70.5%	1×10^{-4}%	98.1%
DCDA	AlexNet	6.74	6.60	5.48
	VGG16	6.75	6.71	4.21
	ResNet50	6.74	6.37	4.19

RBHR: Row Buffer Hit Rate. Access#: Number of DRAM Accesses. BIR: Bank Interleaving Rate. DCDA: DRAM Cycles / DRAM Access

From previous discussion, we know that low RBHR means the row buffers miss/conflict frequently and the average DRAM access time will increase. Low BIR means the same bank-group is accessed frequently, and will cause long average access time. Since DRMap has low RBHR and BIR, its DCDA would certainly become high. For DRMapS, although its RBHR is about the same as DASC, its BIR is much lower than DASC. Thus the DCDA of DRMapS is higher than DASC. Overall, DASC holds the lowest DCDA among different schemes. This provides an obvious proof that our mapping methodology costs less time of DRAM accesses and attains higher DRAM efficiency than other methods.

C. Execution Time of Finding Tiling Factors

In Stage 1 of DASC, we proposed an analytical approach to quickly find proper tilting factors. TABLE V shows the actual execution time (in seconds) spent on finding tiling factors. According to our measurement, for DRMap [9] with step = 2 and 4, the searching cannot be completed within reasonable time. Larger searching steps could shorten the search period. With step=16, DRMap can converge the parameter searching in hundreds to thousands of seconds. Whereas for all the test layers, DASC can find the tiling factors within seconds.

TABLE V. EXECUTION TIME (SECONDS) OF FINDING TILING FACTORS

Model	DASC	DRMap (step=16)	DRMap (step=4)	DRMap (step=2)
AlexNet	0.6~4.2	390~1334	20607~65377	NC
VGG16	1.2~63.6	3562~NC	NC	NC
ResNet50	0.4~9.5	165~1947	8844~NC	NC

NC: Not Completed after running for one day.

D. Supporting Dense CNN

Although DASC is designed to map data of sparse CNN, the flow in Fig. 3 can also handle the dense CNN with very minor adjustment. Stage 1 can bypass the processing of Index and NZ, and only considers the data part. In Stage 2, the scheduling scheme remains unchanged. In Stage 3, the data mapping algorithm can simply ignore the mapping of Index and NZ. When compared with DRMap, our experiments have shown that DASC attains up to 1.47x ,1.80x and 1.85x better DRAM performance for AlexNet, VGG16 and ResNet50 respectively. The performance advantages of DASC are enabled mainly because the exploitation of the bank-group parallelism to shorten the DRAM cycles.

VI. CONCLUSION

Sparse CNNs present substantially smaller models than dense CNNs, however, the compressed data format and irregular accesses of sparse CNN cause challenges in DRAM data mapping. In this paper, we propose DASC, a DRAM data mapping methodology for sparse CNNs. DASC applies efficient data layout and data block schedule to attain good spatial locality and efficient DRAM accesses. The results have demonstrated that DASC decreases the total DRAM latencies and attains an average of 17.1x, 14.3x, and 23.3x better DRAM performance for sparse AlexNet, VGG-16, and ResNet-50 respectively. The analysis of different indicators of DRAM characteristics also shows that DASC poses more efficient DRAM accesses.

ACKNOWLEDGMENT

This work is supported by Ministry of Science and Technology, Taiwan, under grant 110-2224-E-A49-004.

REFERENCES

[1] Bochkovskiy, A., Wang, C., & Liao, H. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. ArXiv, abs/2004.10934

[2] L. Li, M. Xu, X. Wang, L. Jiang and H. Liu, "Attention Based Glaucoma Detection: A Large-Scale Database and CNN Model," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 10563-10572.

[3] T. Chen et al., "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in ASPLOS, 2014

[4] Y. Chen et al., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," IEEE JSSCC, vol. 52, no. 1, pp. 127–138, Jan 2017

[5] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in Proc. of MICRO, 2016.

[6] Song Han, Jeff Pool, John Tran, William Dally, "Learning both Weights and Connections for Efficient Neural Network" in NIPS, 2015

[7] Parashar, Angshuman et al. "SCNN: An accelerator for compressed-sparse convolutional neural networks." 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture, 2017, pp. 27-40.

[8] Bo-Cheng Lai , Jyun-Wei Pan, and Chien-Yu Lin, "Enhancing Utilization of SIMD-Like Accelerator for Sparse Convolutional Neural Networks", IEEE Transactions on Very Large Scale Integration Systems, Volume: 27, Issue: 5, May 2019.

[9] R. V. Wicaksana Putra, M. Abdullah Hanif and M. Shafique, "DRMap: A Generic DRAM Data Mapping Policy for Energy-Efficient Processing of Convolutional Neural Networks," 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1-6.

[10] K. Guo et al., "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 1, pp. 35-47, Jan. 2018.

[11] M. A. Islam, M. Y. Arafath and M. J. Hasan, "Design of DDR4 SDRAM controller," 8th International Conference on Electrical and Computer Engineering, 2014, pp. 148-151.

[12] Micron, "4Gb: x8, x16 Automotive DDR4 SDRAM Features". available:https://www.micron.com//media/client/global/documents/products/data-sheet/dram/ddr4/4gb_auto_ddr4_sdram.pdf

[13] Y. Lee, H. Kim, S. Hong and S. Kim, "Partial Row Activation for Low-Power DRAM System," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 217-228.

[14] W. Shin et al., "Rank-Level Parallelism in DRAM," in IEEE Transactions on Computers, vol. 66, no. 7, pp. 1274-1280, 1 July 2017.

[15] J. Li et al., "Smartshuttle: Optimizing off-chip memory accesses for deep learning accelerators," in Prof. of the Design, Automation Test in Europe Conference Exhibition (DATE), March 2018

[16] Y. Kim, W. Yang and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," in IEEE Computer Architecture Letters, vol. 15, no. 1, pp. 45-49, 1 Jan.-June 2016.

[17] DRAMPower: Open-source DRAM Power & Energy Estimation Tool Karthik Chandrasekar, Christian Weis, Yonghui Li, Sven Goossens, Matthias Jung, Omar Naji, Benny Akesson, Norbert Wehn, and Kees Goossens URL: http://www.drampower.info

[18] A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks," in NIPS, 2012.

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

[20] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi, "Soft Threshold Weight Reparameterization for Learnable Sparsity", Proceedings of the 37 th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

[21] Y. Jia Caffe : A deep learning framework. Available: http://caffe.berkeleyvision.org/

[22] S. V. Lab. (2012). Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) Available:http://www.imagenet.org/challenges/LSVRC/2012/