2025 Design, Automation & Test in Europe Conference (DATE 2025)

SparDR:Accelerating Unstructured Sparse DNN Inference via Dataflow Optimization

Wei Wang, Hongxu Jiang, Runhua Zhang, Yongxiang Cao, Yaochen Han
Beihang University, Beijing, China
{ww_bh, jianghx, rhzhang20, caoyongxiang, hanyc}@buaa.edu.cn

***Abstract*—Unstructured sparsity is becoming a key dimension in exploring the inference efficiency of neural networks. However, its data layout presents irregularity, making it difficult to match the parallel computing mode of hardware, resulting in low computational and memory access efficiency. We have studied this issue and found that the main reason is that existing sparse acceleration libraries and compilers perform sparse matrix multiplication optimization exploration through the splitting and reconstruction of sparse patterns, thus ignoring the acceleration of sparse convolution operations centered on data streams, which may miss some optimization opportunities for sparse operations. In this article, we propose SparDR, a general sparse convolution operation acceleration method centered around data streams. Through novel feature map data stream reconstruction and convolutional kernel data representation, redundant zero value calculations are effectively avoided, addressing efficiency is improved, and memory overhead is reduced. SparDR is based on TVM and allows for automatic scheduling across different hardware configurations. Compared with the current mainstream five methods on four types of hardware, the inference delay acceleration reaches 1.1-12× and the memory usage decreases by 20%.**

***Index Terms*—dataflow, unstructured sparse convolution, scheduling, TVM**

I. Introduction

As deep learning models grow in complexity, sparsification has become one of the crucial methods to enhance the efficiency of neural network inference [1]. Unstructured sparsity, due to its finer granularity in comparison to structured sparsity, achieves higher accuracy. Furthermore, this type of sparsity is a natural characteristic of many neural networks, as it is inspired by biological neural networks, which inherently exhibit unstructured sparsity.

Unfortunately, unstructured sparsity is not yet effectively utilized by hardware to enhance the inference efficiency of neural networks, and the reduction in computational load provided by sparse models does not directly translate into improved performance in model inference. Firstly, sparse representation methods that do not consider data flow overlook the issue of discontinuous memory access, making it difficult to align with the parallel computing patterns of hardware. Even with NVIDIA's highly tailored, manually optimized CUDA sparse library, cuSPARSE [2], the performance still does not match that of the dense computation library CUBLAS when sparsity reaches 95%(Fig. 1). In practical inference tests on a ResNet50 network using the ImageNet dataset(Fig. 2), we found that after weight pruning exceeding 90%, the proportion of zero elements in activations(randomly sampled 10 convolutional layers) only ranged from 0.35 to 0.65 (matrices are generally termed sparse only when the zero element ratio is higher than 0.7). Secondly, current sparse acceleration libraries and compilers often concentrate on the disassembly and reconstruction of sparse patterns. However, this method has significant limitations as it requires high demands for non-zero topology structures, and pattern recognition and adaptation itself are time-consuming and laborious tasks. In addition, the general methods focus on Sparse general matrix-matrix multiplication (spGEMM) for scalability. Although convolutions can be transformed into matrix multiplications through steps like Im2col [3], this transformation is originally intended to utilize mature kernel implementations in the matrix optimization field, which are not suitable under sparse conditions. On the contrary, these methods may overlook unique optimization opportunities related to the data flow inherent in sparse convolution operations. Convolution operations are inherently performed within local regions, allowing for efficient data reuse and cache utilization [4]. However, once converted into matrix multiplication, this locality is lost because matrix multiplication involves more extensive data access, which could potentially reduce the efficiency of data access caching. This situation is particularly pronounced in sparse cases.

To address the above issues, this work proposes SparDR, a dataflow-centric, unconstrained sparse convolution acceleration method. SparDR reconstructs feature map data to exploit locality for tighter data organization, effectively sidestepping unnecessary zero-value computations and alleviating the issue of disjointed memory access. It also enhances data reuse and diminishes memory costs. Shifting focus away from the topology of non-zero values, SparDR employs a strategy during the compilation stage that zeroes in on the initial positions of receptive fields within convolutions. This approach circumvents the addressing computations typically associated with sparse operations. Moreover, by implementing bucket sorting, SparDR addresses load imbalance within sparse computations, thereby facilitating additional acceleration of inference processes.

The main contributions of this work are summarized as follows:

- **Method.** We introduce memory continuous and efficient feature map reconstruction, and sparse weight representation completed offline, to accelerate inference.
- **Framwork.** We have implemented SparDR based on TVM

This work is supported by the National Key Research and Development Program of China (No.2022YFB4501600). Hongxu Jiang is the corresponding author and Runhua Zhang is the co-first author.

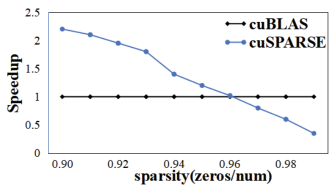


Fig. 1. Speed comparison.

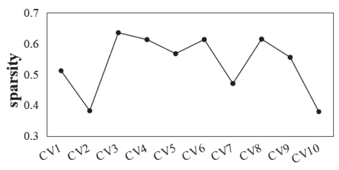


Fig. 2. Sparsity of activation.

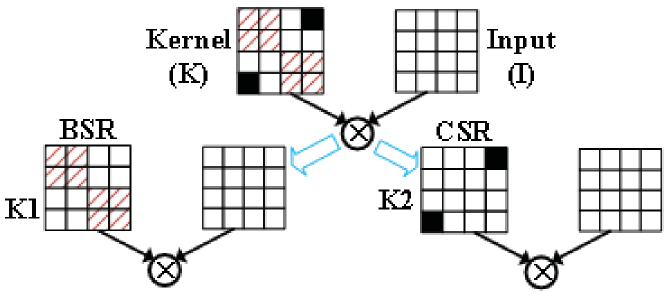


Fig. 3. Disassembly and reconstruction of sparse patterns.

[5] , including redefining sparse convolution operators, improving sparse scheduling using predefined sketches, etc., to achieve multi hardware adaptation.

- **Evaluation.** We conduct comprehensive experiments to compare SparDR with multiple baselines on different platforms (i.e., CUDA GPU, ARM CPU,Intel CPU). Our evaluation shows that SparDR achieves an average acceleration of 7.4× in model inference latency with less memory consumption, compared to the five mainstream solutions.
- **Open-Source.** We have made our code open-sourced at https://github.com/wang-hub/SparDR.

II. Background

A. *Sparsity on DNN models.*

The sparsity of deep learning models is a vibrant and extensively researched topic. Beyond the inherent sparsity present in models in most scenarios, model pruning represents a significant source of sparsity. Structured pruning compresses models to achieve accelerated inference by trimming unimportant convolutional kernels—including entire kernels or certain unimportant channels within them—or network layers. However, to maintain the network's accuracy, the pruning rate is often not significantly high. In contrast, unstructured pruning reduces the network size by eliminating unimportant neurons(i.e., parameters within convolutional kernels), which allows for a finer granularity of pruning. Compared to structured pruning, it can significantly reduce the model size while retaining the network's original accuracy. Typical unstructured model pruning methods can remove over 90% of the model's parameters with virtually no loss in accuracy [6].

The precision loss after unstructured pruning is less noticeable. However, due to the random nature of the pruned convolutional kernel elements, it results in irregular patterns of parameters, creating irregular sparsity in the model. When deploying models, this irregular sparsity hampers hardware platforms from executing efficient parallel acceleration.

B. *Sparse Acceleration Exploration in DNN Framework and Compiler.*

Deep Neural Network (DNN) inference is generally conducted through two primary modalities: Deep Learning Acceleration Frameworks and the more contemporary emergence of Deep Learning Compilers.

Universal DNN inference frameworks, such as PyTorch [7], utilize highly customized sparse acceleration libraries provided by vendors—like cuSPARSE [2]—to perform inference on sparse models. However, the efficacy of this method is profoundly contingent on the performance of the supplier's acceleration libraries, which is often suboptimal, as evidenced in Fig. 1. Furthermore, it has been observed that during actual inference processes, even when the parameter pruning rate exceeds 90%, the sparsity of the intermediate sparse process feature maps does not reach 0.7—a common threshold for determining whether a matrix is sparse or dense, as shown in Fig. 2. Consequently, it is not feasible to simplistically treat convolution operations on sparse models as mere interactions between two sparse matrices.

Deep Learning Compilers, such as the Tensor Virtual Machine (TVM) [5], represent an innovative category of deep learning acceleration frameworks. These compilers transform models from various frameworks into a unified intermediate representation, which they then optimize and deploy across different hardware targets. However, these compilers are primarily designed for dense networks and lack robust support for sparse networks. Attempts to incorporate sparse support often concentrate on the disassembly and reconstruction of sparse patterns, such as seeking a performance-optimized Blocked Sparse Row (BSR) structure within the Compressed Sparse Row (CSR) representation [8], as illustrated in Fig. 3. SparTA [9] and SparseTIR [10] are both based on this idea, however, these methods demand a high level of topological sophistication in the arrangement of non-zero values and are both time-consuming and labor-intensive in terms of pattern adaptation.

Moreover, to achieve scalability, the aforementioned methods predominantly target sparse matrix multiplication operations. However, converting all sparse convolution computations into matrix multiplications does not necessarily constitute the optimal implementation strategy. This approach may indeed fail to exploit distinctive optimization opportunities related to the data flow characteristics inherent in sparse convolution operations.

III. Design of SparDR

A. *Overview of the SparDR*

Fig. 4 illustrates the overall architecture of SparDR designed for inference. SparDR is implemented on TVM [5], facilitating compatibility with multiple model input frameworks and various backend adaptations. At the heart of SparDR's operations lies the restructurion of feature dataflow and the representation of sparse kernel. The implementation core encompasses specific operator realizations and transformations of the Relay graph encapsulated into Pass [11] (a single traversal and operation of the Intermediate Representation), as well as optimization tuning

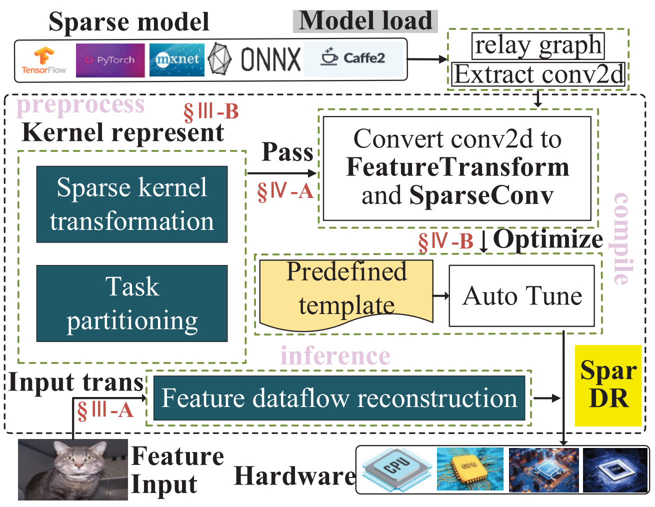


Fig. 4. Overview of SparDR design.

TABLE I
NOTATIONS USED IN THIS PAPER

Not.	Description of the notation
I	Input tensor($C_i I_h I_w$) of conv
L	Intermediate results of I transformation
K	Kernel tensor($C_o C_i K_h K_w$) of conv
IM	Intermediate results of K transformation
O	Output tensor($C_o O_h O_w$) of conv
$A_{(i,j)}$	The element at index [i, j] of matrix A
$A_{[i:]}$	The i-th row of the matrix A
$A_{[:j]}$	The j-th col of the matrix A
$A_{[i:a,j:b]}$	The submatrix of A from indices [i, j] to [a, b]
p, s	padding, stride

across multiple platforms utilizing predefined sketch to refine sparse computations.

The actual inference process is divided into three main stages: preprocessing, compilation, and inference. Kernel representation is performed in the preprocessing stage. Following this, the transformation and optimization of the Relay graph occur in the compilation stage. Both of these stages can be completed offline. The dataflow restructuring of the input feature maps happens in the inference stage. To better illustrate the connections among the methods used in this paper, we will first introduce the dataflow restructuring and the kernel representation, and then describe the implementation process. Table I provides an overview of the basic symbols and their descriptions used throughout this paper.

B. Feature Dataflow Reconstruction

1) Data Aggregation: We transform the input I into a new two-dimensional matrix form L. The commonly used Im2col method gathers the receptive field ($K_h K_w$ data) for each element in the output into a single row. Similarly, we apply this concept to sparse convolutions, with the distinction of gathering the local window of weights ($O_h O_w$ data) each time, with the entire process being accomplished via a sliding window

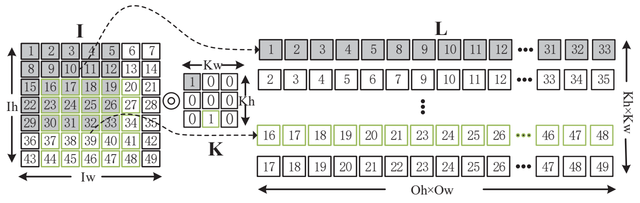


Fig. 5. Data aggregation.

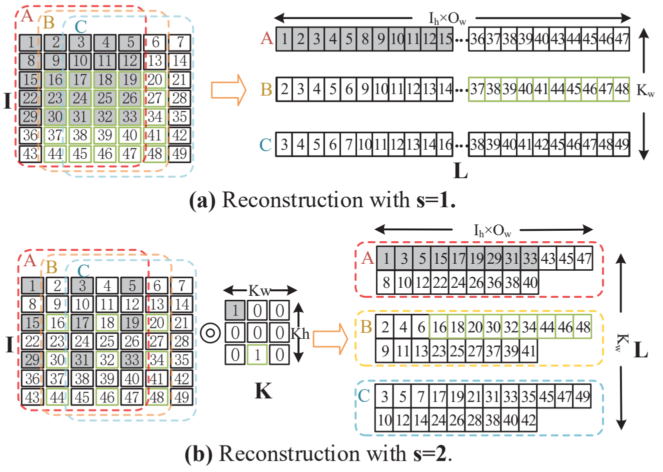


Fig. 6. Vertical data reuse.

technique. As depicted in Fig. 5, the gray area($I_{[0:6,0:6]}$) represents the data window required for $K_{(0,0)}$, which is aggregated into $L_{[0:]}$. Following this, the sliding window transformation is conducted according to the stride. The green area indicates the data window required for $K_{(2,1)}$, which will be aggregated into $L_{[8:]}$. When a 7x7 matrix is fully transformed, the size of result L is 9x25($K_h K_w$×$O_h O_w$), consistent with Im2col.

2) Vertical Data Reuse: Following the gathering process, numerous redundant values emerge, and we introduce vertical data reuse. As shown in Fig. 6, after completing the data required for $K_{(0,0)}$ (gray area), the process continues with data in the vertical direction ($A(I_{[0:7,0:5]})$, rearranging it into $L_{[0:]}$. In other words, we redefine the sliding window size (from $O_h O_w$ data to $I_h O_w$ data). When stride (s) is equal to 1, the next partition, B($I_{[0:7,1:6]}$), is transformed. Continuing this process, in this example, will eventually form three sets of reconstructed data $\{A, B, C\}$, with a data storage capacity of 3x35 (K_wx$I_h O_w$), which represents a 53% reduction compared to the data gathering method. In practice, the ratio R between the two is shown in (1). When s is not equal to 1, the concept continues to leverage data aggregation and vertical reuse. Just the local window required for the weight moves according to s, leading to non-continuous rows and columns. As shown in Fig. 6(b). The gray area is still the required data window for $K_{(0,0)}$, but it is no longer continuous.

$$R = \frac{(i_h - k_h)\left(\frac{k_h}{s} - 1\right)}{i_h} + 1 \qquad (1)$$

For multi-channel, we employ channel-first strategy. Algorithm 1 delineates the address mapping relationship before and after transformation, highlighting that there are no write

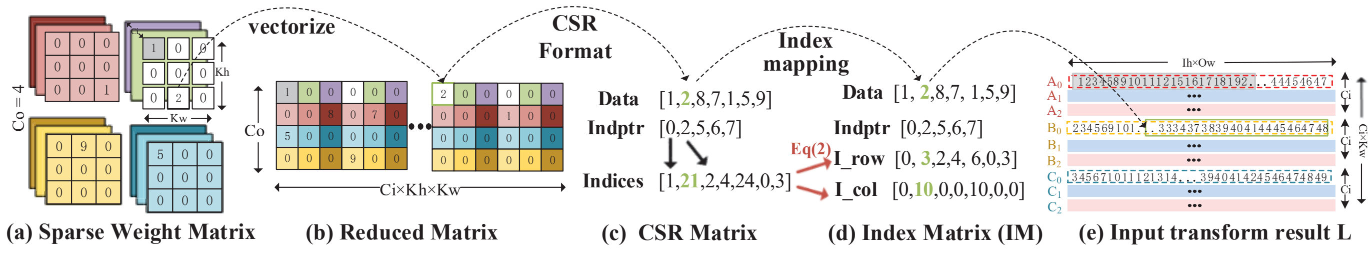


Fig. 7. Sparse kernel representation.

conflicts between different rows, enabling complete parallelism. It is noteworthy that all data from I is essential. There is no scenario in which sparse weights guide the reordering to alleviate the computational burden. Each element within I must be multiplied by the weights at the identical location across all output channels, such as, the element $I_{[0,0,0]}$ must not only be multiplied with the first convolution kernel $K_{[0,0,0,0]}$ to obtain the output $O_{[0,0,0]}$, but also be multiplied with the second kernel $K_{[1,0,0,0]}$ to get the $O_{[1,0,0]}$, and this process continues until the calculations for all channels are completed. Convolutional operations typically involve dozens to hundreds of output channels. Even when model sparsity exceeds 90%, it is uncommon to encounter a situation where all output channels possess zero weights at the same location without guidance.

Algorithm 1 Multi-channel Feature Dataflow Reconstruction

Input: I:(input, $c_i i_h i_w$), kernel size($c_o c_i k_h k_w$), p:padding
Output: L:Conversion results, as $c_i k_w * o_h(i_h + 2 * p)$ tensor

```
1: for $row = 0$ to $rows : k_w * c_i$ parallel do
2:     $c,\ offsize \leftarrow row \bmod c_i,\ row \bmod k_w / c_i$
3:     for $col = 0$ to $cols : o_h * (i_h + 2 * p)$ do
4:         $h \leftarrow col / o_h$
5:         $w \leftarrow col \bmod o_h + offsize$
6:         if $h \geq p\ \&\ h - p \leq i_h\ \&\ w \geq p\ \&\ w - p \leq i_h$ then
7:             $L_{row,col} \leftarrow I_{c,h,w}$
8:         else
9:             $L_{row,col} \leftarrow 0$
```

C. Kernel Represent

1) Index Mapping: We initially reduces the kernel to a two-dimensional format and then employs the CSR representation to effectively minimize the storage of redundant zero values within the kernel, both of which can be readily implemented. The innovation of this paper lies in integrating previous work(III-B), utilizing the mapping relationship between the CSR matrix and L. We keep the focus on the convolution's receptive field, aligning the kernel with the corresponding positions on the input, thereby enhancing the addressing efficiency during computation. The specific process is depicted in Fig. 7. During the reduction of the kernel's matrix 7(a) into a two-dimensional matrix 7(b), the data for each convolution kernel is organized into a single row, following the channel-prior principle, and the sequence is Height-Width-Channel (HWC). The CSR representation is shown in 7(c). Finally, to facilitate

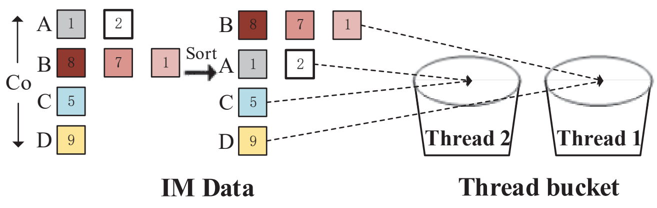


Fig. 8. The bucket sorting process.

efficient addressing during convolution calculations, we utilize (2) to map the CSR indices into an index matrix(IM) 7(d), which is then mapped with L 7(e). In the IM 7(d), the I_{row} and I_{col} values indicate the starting positions in L for the local windows corresponding to non-zero weight. For example, the position (3,10) corresponds to the green box in 7(e).

$$\begin{cases} I_{row} = Indices \mod (I_h \times O_w) \\ I_{col} = \dfrac{Indices}{I_h \times O_w} \end{cases} \tag{2}$$

2) Task partitioning: For the computation tasks involving non-zero values within the kernel(each row in the matrix of Fig. 7(b)), We adopts inter kernel task partitioning, where all non-zero value computing tasks are assigned to a single computing unit to achieve parallelism. utilizing a simple bucket sorting approach to address the issue of uneven task load distribution among multiple threads or computing units. Initially, each row of the sparsely represented matrix is sorted in descending order by the number of non-zero data points, and the number of threads or parallel computations supported by the hardware is considered as buckets. Following this, the sorted workload is allocated from largest to smallest into the bucket with the least load. As illustrated in Fig. 8, {A, B, C, D} represent different tasks from Fig. 7(b), with their workloads measured by the number of non-zero values being {2, 3, 1, 1} respectively. After sorting in descending order, the sequence becomes {B, A, C, D}. Assuming the hardware supports dual-threading (two buckets), the task with the largest workload, B, is first added to thread 1 bucket, which currently has the least workload. Subsequently, the task A with the current largest workload is allocated to thread 2 bucket. This process continues until the last task D is placed into a bucket, marking the completion of task allocation.

The content of this chapter is conducted during the preprocessing stage, despite the high time complexity of sorting operations and the intricate structure of thread bucket priority queues, these processes can be entirely completed offline. As a result, they do not cause any additional operations or time delays during the inference phase.

IV. Implementation

A. SparseConv Trans Pass

This paper implements SparDR on TVM, first addressing the integration of the framework to accommodate inputs from various types of models. The framework integration process includes operator definition and computation graph transformation. As defined in the previous chapterIII, we have defined data processing operators (KernelTransform), feature map transformation operators (FeatureTransform), and sparse convolution operators (SparseConv). Among these, the KernelTransform operator is invoked during the preprocessing stage when the model is loaded. The FeatureTransform operator and the SparseConv operator, after tensor kernel loop matching (a requirement for operator writing in TVM), await scheduling optimization during the compilation stage. After completing the operator definitions, by implementing a traversal function (ExprFunctor), after the model is read into the form of TVM's Relay Graph, the corresponding operators are replaced with SparseConv operators, and FeatureTransform operators are inserted. We integrate the above process into a Pass [11] to achieve convenient and user-friendly calling

B. Auto Optimization

SparDR then addresses the optimization challenge for sparsity to adapt to the deployment on various types of platforms. The sparse optimization process includes constructing an optimization space and executing scheduling optimization. We accomplish the construction of the optimization space through predefined sketch and then invoke TVM's scheduling optimization algorithms to optimize sparse operators, aiming to match the optimal code implementation for different hardware backends. TVM primarily forms the optimization space by splitting loop axes (the reason for tensor kernel loop matching in Chapter IV-A), searching for the optimal implementation of the code within the optimization space, with the boundaries of the loop axes fixed to determine the search space and strategy. If there are composite indices (such as a[b[i]], which exists in all methods of compressed sparse matrices), it leads to undefined loop boundaries, making optimization impossible. To fix the loop boundaries, the predefined sketch optimization space construction is executed. In the sketch, the application function is used to determine the optimization space. First, the loop axes used in the sparse operator computation are extracted. According to the sparsity of the convolution kernel, the upper and lower bounds and the splitting granularity of the loop axes are defined (using TVM's default splitting granularity). Then, the Split statement is used to complete the splitting of the axes, and the Reorder statement is used to reconstruct the axes, forming a scheduling space for sparse operators. Finally, a scheduling algorithm is called (using TVM's built-in XGBoost) to optimize sparse operators, matching the optimal code implementation for different hardware backends.

TABLE II
CONV BENCHMARK

NAME	INPUT $C_i I_h I_w$	KERNEL $C_0 C_i K_h K_w$	P,S	SPARSITY
CV1	179,14,14	179,179,3,3	1,1	0.827797
CV2	179,14,14	716,179,1,1	0,1	0.831005
CV3	179,28,28	179,179,3,3	1,2	0.827987
CV4	2048,7,7	358,2048,1,1	0,1	0.881186
CV5	358,28,28	716,358,1,1	0,2	0.829097
CV6	358,7,7	2048,358,1,1	0,1	0.881133
CV7	44,56,56	44,44,1,1	0,1	0.871384
CV8	716,14,14	2048,716,1,1	0,2	0.879547
CV9	89,28,28	89,89,3,3	1,1	0.832793
CV10	512,14,14	512,512,3,3	1,1	0.900000
CV11	192,40,40	192,192,3,3	1,1	0.890501
CV12	288,20,20	288,288,3,3	1,1	0.878399
CV13	96,80,80	96,96,3,3	1,1	0.928204

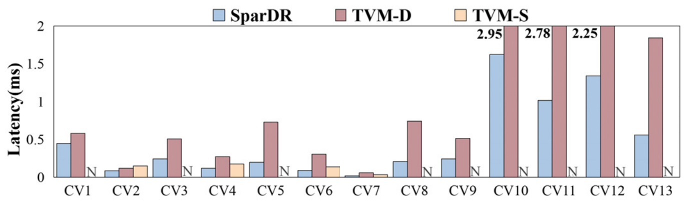
V. Evaluation

A. Experimental Settings

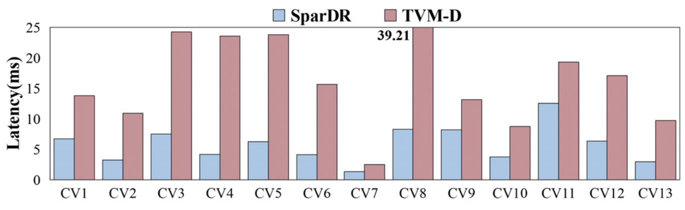
To verify the effectiveness, We tested unstructured sparse ResNet50, VGG19, and YOLOv8 models from SparseZoo [12]. After training on the ImageNet and COCO datasets, the overall sparsity was between 70% and 90%, with accuracy approximating that of the original models. Table II presents a comprehensive benchmark test set consisting of 13 convolutional layers, including commonly used convolution sizes of 3×3 and 1×1, various convolution parameter arrangements (p={0,1}, s={1,2}), and multiple output channel configurations. We evaluated the effectiveness of SparDR on CPUs on Intel Xeon CPU E5-2620 (X86) and Raspberry Pi 3B (ARM), and on GPUs on RTX3090 and V100. In addition to SparDR, we also covered mature methods integrated into TVM [5], common sparse library(cuSPARSE) [2], general-purpose deep learning framework(PyTorch) [7], and deep learning sparse compiler(SparTA) [9]. The tuning rounds requiring search were all set to 1000 (the number recommended by TVM).

B. Experimental Results

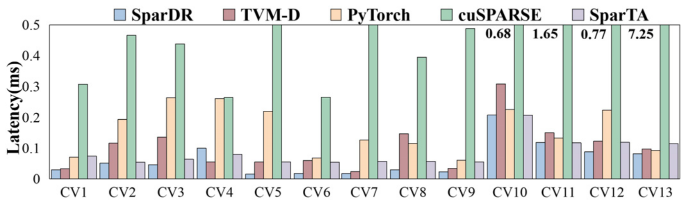
1) Evaluation of latency: Referring to Fig. 9, (a) demonstrates that on x86 architecture CPU, SparDR achieves an acceleration of 1.3-3.6× compared to TVM-D. In the case of 1x1 kernels, the acceleration over TVM-S exceeds 1.4×. (b) illustrates the results on ARM architecture CPU, showing that SparDR achieves more than 1.5× acceleration over TVM-D. In the case of 1x1 kernels with a larger number of channels, SparDR can achieve up to 3× acceleration, with the highest (CV4) reaching 5.6× acceleration. (c) presents the results on RTX3090. In the case of 1x1 kernels where the number of input channels is much smaller than the number of output channels (CV4), SparDR slightly underperforms compared to TVM-D and sparTA. However, in other cases, SparDR accelerates 1.1-5× over TVM-D and 1-3.6× over sparTA. In all cases, the acceleration over PyTorch is more than 1.1×, and over cuSPARSE, the highest acceleration (CV5) can reach 33×. (d) shows the results on V100. In the case of CV4, SparDR's performance is still not ideal, which is in contrast to the results



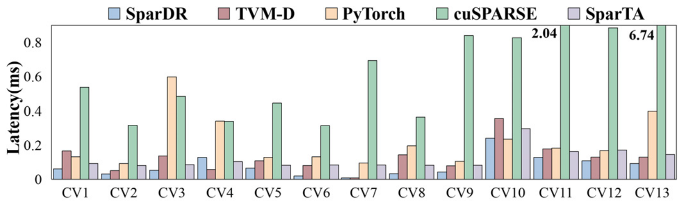
(a) Runtime on X86 CPU.



(b) Runtime on ARM CPU.



(c) Runtime on RTX3090 GPU.



(d) Runtime on v100 GPU.

Fig. 9. Performance of various convolution algorithms.

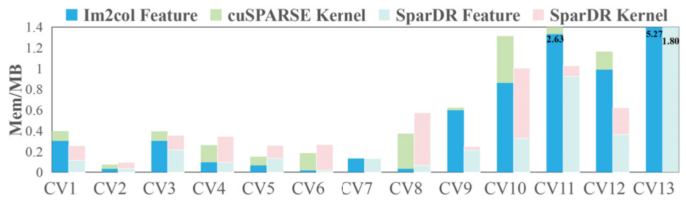


Fig. 10. Memory usage.

on CPUs but is understandable. A large number of inputs leads to more threads, and a small number of input channels causes greater data conflict, cache's constant failure to hit, which is fatal to GPU parallelism. In other cases, SparDR can achieve 1-4.4× acceleration over TVM-D, 1-12× acceleration over PyTorch, more than 3× acceleration over cuSPARSE, and also 1.2× acceleration over sparTA.

Convolution is the most crucial component in the ResNet network. To approximately evaluate the overall inference time, we summarized the time results for all pruned convolutional layers in ResNet50. As shown in Table III, SparDR achieves an acceleration of 1.53× to 2.44× over TVM on four different platforms, and compared with PyTorch, it accelerates by 3.10× to 4.11× on GPUs.

2) Evaluation on memory: In addition to runtime, memory consumption is also an important factor affecting inference. In the case of 1x1 convolution, traditional feature map rearrangement does not increase additional memory, so SparDR's memory usage may be slightly higher, as shown in cv2, cv4-cv8 in Fig. 10. However, in non-1x1 situations, the additional memory usage of SparDR decreases significantly. The larger the feature map size, the more the memory decreases (for example, CV11, CV13, with feature map sizes of 40*40 and 80*80, respectively, memory usage decreases by 62%, 66%). Therefore, in the sparsified ResNet50 network, where non-1x1 sized convolutions in sparse convolutions account for only 38%, the total memory usage decreases from the original 17.05MB to 14.26MB, a reduction of 16% overall.

TABLE III
RESNET50 RUNTIME

	RUNTIME (ms)			RATIO		
	SparDR	TVM	PyTorch	SparDR	TVM	PyTorch
X86	9.37	18.64	NA	1.0	**1.99**	NA
ARM	299.01	730.15	NA	1.0	**2.44**	NA
RTX3090	2.12	3.32	6.58	1.0	**1.57**	**3.10**
V100	2.58	3.94	10.62	1.0	**1.53**	**4.11**

VI. RELATED WORK

Several related works have been proposed in the field of utilizing sparsity to accelerate DNN inference. cuSPARSE [2] implements platform-specific sparse convolution computations by calling the Im2col [3] and GEMM methods. This approach heavily relies on the performance of computing libraries, and optimizing these libraries requires a significant amount of human effort, as well as demanding programming expertise from developers in hardware familiarity. TVM [5] is an end-to-end machine learning compilation framework and it has also attracted a lot of sparse acceleration work. Liao and et al. [13] provides a sparse convolution flow with weight pruning and replaces conv2d into im2col_transform operator and dense operator. Ye [10] propose SparseTIR, a sparse tensor compilation abstraction that offers composable formats and composable transformations for deep learning work loads. However, they still attempt to schedule sparse computation using the mindset of dense computation and have low acceleration efficiency. SparTA [9] proposes abstractions for model sparsity; utilizing sparse propagation and pattern splitting for inference acceleration. However, pattern recognition is time-consuming and laborious, and its annotation is still dense.

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose SparDR, a sparse convolution acceleration method that accelerates inference by dataflow reconstruction. We achieve multi platform acceleration by redesigning sparse operations and compatibility with TVM automatic scheduling.We have evaluated SparDR through multiple mature pruning models on multiple platforms.SparDR is up to 0.5-5.1× faster than TVM [5], up to 1.1-14.7× faster than the PyTorch [7], over 2.6× compared to the cuSPARSE library [2], and 0.8-11.1× compared to SparTA [9].

In the future, we will attempt to apply SparDR on NPU and FPGA platforms. In addition, we plan to incorporate the determination of sparse thresholds into the tuning steps.

References

[1] oshida T, Ohki K. Natural images are reliably represented by sparse and variable populations of neurons in visual cortex[J]. Nature communications, 2020, 11(1): 872.

[2] The api reference guide for cusparse, the cuda sparse matrix library, 2021. [Online]. Available: https://docs.nvidia.com/cuda/cusparse/index.html

[3] Chellapilla K, Puri S, Simard P. High performance convolutional neural networks for document processing[C]//Tenth international workshop on frontiers in handwriting recognition. Suvisoft, 2006.

[4] Zhang R, Jiang H, Geng J, et al. A high-performance dataflow-centric optimization framework for deep learning inference on the edge[J]. Journal of Systems Architecture, 2024, 152: 103180.

[5] Chen T, Moreau T, Jiang Z, et al. TVM: An automated End-to-End optimizing compiler for deep learning[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 578-594.

[6] Peste A, Vladu A, Kurtic E, et al. Cram: A compression-aware minimizer[J]. arxiv preprint arxiv:2207.14200, 2022.

[7] Imambi S, Prakash K B, Kanagachidambaresan G P T. In Programming with TensorFlow: Solution for Edge Computing Applications[J]. 2021.

[8] Tang A, Quan P, Niu L, et al. A survey for sparse regularization based compression methods[J]. Annals of Data Science, 2022, 9(4): 695-722.

[9] Zheng N, Lin B, Zhang Q, et al. SparTA:Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute[C]//16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022: 213-232.

[10] Ye Z, Lai R, Shao J, et al. Sparsetir: Composable abstractions for sparse compilation in deep learning[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. 2023: 660-678.

[11] Lattner C, Adve V. A compilation framework for lifelong program analysis and transformation[C]//CGO. 2003, 4: 75.

[12] Kurtz M, Kopinsky J, Gelashvili R, et al. Inducing and exploiting activation sparsity for fast inference on deep neural networks[C]//International Conference on Machine Learning. PMLR, 2020: 5533-5543.

[13] Liao H H, Lee C L, Lee J K, et al. Support convolution of CNN with compression sparse matrix multiplication flow in TVM[C]//50th international conference on parallel processing workshop. 2021: 1-7.