RFC-HyPGCN: A Runtime Sparse Feature Compress Accelerator for Skeleton-Based GCNs Action Recognition Model with Hybrid Pruning

***Abstract*—Skeleton-based Graph Convolutional Networks (GCNs) models for action recognition have achieved excellent prediction accuracy in the field. However, limited by large model and computation complexity, GCNs for action recognition like 2s-AGCN have insufficient power-efficiency and throughput on GPU. Thus, the demand of model reduction and hardware acceleration for low-power GCNs action recognition application becomes continuously higher.**

To address challenges above, this paper proposes a runtime sparse feature compress accelerator with hybrid pruning method: RFC-HyPGCN. First, this method skips both graph and spatial convolution workloads by reorganizing the multiplication order. Following spatial convolutions channel-pruning dataflow, a coarse-grained pruning method on temporal filters is designed, together with sampling-like fine-grained pruning on time dimension. Later, we come up with an architecture where all convolutional layers are mapped on chip to pursue high throughput. To further reduce storage resource utilization, online sparse feature compress format is put forward. Features are divided and encoded into several banks according to presented format, then bank storage is split into depth-variable mini-banks. Furthermore, this work applies quantization, input-skipping and intra-PE dynamic data scheduling to accelerate the model. In experiments, proposed pruning method is conducted on 2s-AGCN, acquiring 3.0x-8.4x model compression ratio and 73.20% graph-skipping efficiency with balancing weight pruning. Implemented on Xilinx XCKU-115 FPGA, the proposed architecture has the peak performance of 1142 GOP/s and achieves up to 9.19x and 3.91x speedup over high-end GPU NVIDIA 2080Ti and NVIDIA V100, respectively. Compared with latest accelerator for action recognition GCNs models, our design reaches 22.9x speedup and 28.93% improvement on DSP efficiency.

***Index Terms*—Graph Neural Network, Action Recognition, Hybrid Pruning, Sparse Data Compress, Hardware Accelerator, Field-programmable Gate Array (FPGA)**

I. Introduction

Action recognition based on deep learning has the great potential to be applied in kindergartens, hospitals and stadiums to prevent danger motions. Skeleton-based graph convolutional networks (GCNs) methods have achieved state-of-the-art (SOTA) prediction accuracy in the field [1] [2] [3]. Mature pose estimation algorithms extract human skeletons from video stream with real-time speed, for example, OpenPose [4] and Alphapose [5]. GCNs action recognition models and pose estimation models thus can be combined into an end-to-end system.

Despite skeleton-based GCNs having great advantages, several problems limit their applications in expected scenarios. Firstly, intensive computation and large network architectures are embedded in skeleton-based GCNs, causing great computing cost on GPUs. MobilePose [6] can produce human skeletons on mobile platform Snapdragon 845 with 60 fps and 44.4 fps/Watt, while 2s-AGCN model merely has a performance of 28 fps and 0.11fps/Watt on NVIDIA 2080Ti GPU. The computing speed and power-consumption's gap indicates a great importance on accelerating GCNs action recognition algorithms. Secondly, the expected application environment of action recognition models poses stringent constraints on power-consumption and throughput. However, the high-performance GPU cannot meet the power-efficiency demand.

Network pruning and graph sparsification [7] [8] are two effective methods to relieve model complexity. However, these methods are unsuitable for skeleton-based GCNs. There are two reasons. (i) *Dataflow is transformed*: Graph computation changes the convolution dataflow. When being conducted on different dataflows, traditional pruning methods for CNNs only skip useless computing in convolution but may not work on graph task. (ii) *Skeleton-relationship graph is unchangeable and sensitive.* Some works use pooling [7] or graph sparsification [8] to drop unimportant edges and points in graph. However, the human skeleton graph cannot be modified for bones connection being unchangeable. Particularly, there are learnable hidden information graph [1] [9] which lacks sparsity in some GCNs models . The subtle elements in such graph are proved to be positively associated with prediction performance. For instance, in 2s-AGCN model, the prediction accuracy decreases by 2.3% without learnable matrix [9]. Although Ding et al. [10] present a FPGA-based work on accelerating ST-GCN, a smaller action recognition GCNs, their work falls short on more complex models for (i) They do not prune the target model. (ii) They choose to compress sparse static skeleton graph, while skeleton relationship matrix in some models can be dense. (iii) Only sparsity graph is optimized for computation, while feature sparsity is not utilized. (iv) Their single-PE design cannot meet performance requirement of application scenarios.

For these reasons, efficient pruning methods together with specific accelerator designs are urgently required to accelerate GCNs action recognition workloads. To this end, we present RFC-HyPGCN: a runtime sparse feature compress accelerator for skeleton-based GCNs action recognition model with hybrid pruning in this paper.

A hybrid GCNs' pruning method is proposed, which can

reduce convolutional parameters and skip graph computation efficiently. We reorganize dataflow by changing the multiplication order of graph workloads and spatial convolution. With new dataflow, a group of graph computation and spatial convolution are skipped if the corresponding parameter is pruned as zero. As to the temporal convolution, mixed-grained pruning method is elaborately designed. Fine-grained pruning operation can be dealt as sampling in time series, while coarse-grained pruning is decided by spatial convolution's pruned dataflow. The experiments demonstrate that in most cases, better prediction accuracy and more balanced pruning can be possessed by our model compared with conventional methods. Additionally, quantization and input-skipping are applied on software level, which are common means to accelerate neural networks.

We also design an application-specific architecture. Ten convolution blocks are mapped on FPGA, which is the platform widely used for speeding up deep neural networks. Different from previous works, in our layer-pipelined architecture, challenges are not only reflected on different kinds of sparse tasks, but also on how to efficiently store sparse intermediate results on chip. Common compact formats like compressed sparse column format (CSC) is not the optimal resolution due its irregular memory access and extra encoding/decoding cost. To address these challenges, our sparse-degree-based runtime sparse feature compress method is proposed, which splits data encoding/decoding and corresponding storage into fine-grained bank and mini-bank. Finally, dynamic data scheduling is applied for intra process elements (PE) to decrease the utilization of DSPs.

In summary, the contributions in this paper are:

- We propose a hybrid pruning method on 2s-AGCN model, which contains dataflow reorganization and mix-grained pruning method. The experiments show that our method is better than conventional pruning on computation-skipping and prediction accuracy.
- A co-designed architecture is implemented, including runtime sparse vector compress storage and dynamic data scheduling. The proposed online data compact format reduces the utilization of BRAM blocks as well as keeping regularity of data-access. Dynamic data scheduling saves DSPs and raises DSPs working efficiency with tiny delay.
- Our design is implemented on Xilinx XCKU-115 FPGA platforms with 172 MHz. It achieves 2.61x-9.19x accelerating ratio compared with NVIDIA 2080Ti and 1.36x-3.91x with NVIDIA V100. On contrast to similar work [10], ours exceeds on both peak performance and DSPs efficiency. It has the potential to apply in end-to-end and low-power real time environments.

II. Background on 2s-AGCN Model

The skeleton-based action recognition GCN models regard human skeleton features as input and predict human action , like waving and drinking. Several human skeleton datasets have been proposed, for example NTU-RGB+D [11] and Kinetics [12]. Our experiments on 2s-AGCN are trained and tested on NTU-RGB+D. There are ten convolutional blocks and one fully-connected (FC) layer in 2s-AGCN model. As shown in the left picture in Fig. 1, the computation in each block can be divided into five phases: graph computation, self-similarity computation, spatial convolution, temporal convolution and shortcut connection. Batch-normalization and ReLU activation follow behind each convolution operations. With network going deeper, more channels are stacked on feature. The right picture of Fig. 1 illustrates this tendency in data dimension.

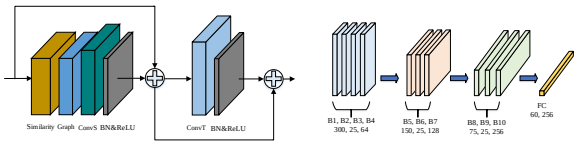

Fig. 1. Left: Structure of the basic convolutional block. ConvS stands for spatial convolution and ConvT stands for temporal convolution. Right: Variance of the feature dimension. There are 25 key joints in human skeleton and 300 skeleton vectors in the original input feature.

In each layer, three different graphs are involved in computation: A_k, B_k and C_k, k is explained in (2). The first part A_k is the static human skeleton graph, the second part B_k is a learnable skeleton connection graph and C_k is a data-dependent graph generated from self-similarity process. Elements in B_k are trained to indicate hidden relationships between joints and bones. Unlike static graph A_k, B_k is dense and sensitive to numerical changes. In [9], C_k is produced via (1), where high-dimension tensor transposition and multiplication are conducted on input feature. W_θ represents similarity coefficient. To sum up, the computation of graph and spatial convolution can be described as (2). $\bigotimes$ represents convolution operation, K_ν denotes the neighbour size of the graph computation and is set to 3 in the 2s-AGCN model. The kernel size of spatial convolutions weight W_k is set to 1.

$$C_k = softmax(f_{in}^T W_\theta f_{in}) \tag{1}$$

$$f_{out} = \sum_k^{k_\nu} f_{in}(A_k + B_k + C_k) \bigotimes W_k \tag{2}$$

Different from A_k and B_k which are determined before inference, C_k relies on input feature, thus needs runtime computing for each prediction. Table. I demonstrates the computing cost of self-similarity. The running performance of 2s-AGCN with and without C_k are tested on NVIDIA V100. At the cost of computing complexity and longer time-delay, C_k only elevates prediction accuracy by 0.3%. From the view of software-hardware co-design, dropping C_k graph is a reasonable trade-off for workload reduction. Following the spatial convolution, temporal convolutional layer is set at the end of each convolutional block. With kernel size of 9×1, temporal convolution extracts information from nine skeleton vectors in time order. Despite the insertion of the graph computation, temporal convolution layer in block l can still be seen as the leading neighbour of spatial convolution

TABLE I
MODEL PERFORMANCE WITH(W/) AND WITHOUT(W/O)C_k.

	accuracy	throughput	power efficiency
2sAGCN(w/C)	93.70%	69.38 fps	0.28 fsp/watt
2sAGCN(w/oC)	93.40%	98.87 fps	0.40 fps/watt

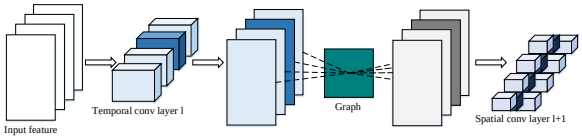

Fig. 2. The neighbour connection between temporal convolutional output and spatial convolutional input.

layer in next block because graph computation does not change temporal convolutional result along its output-channel dimension, and spatial convolution operates indirectly on temporal convolution's output in block $l+1$ [13]. For above reasons, the connection shown in Fig. 2 guides us to conduct coarse-grained pruning on temporal convolutional filters.

III. RELATED WORK

CNNs Accelerators on FPGA. Works on FPGA-based acceleration of sparse CNNs can be categorized by different pruning granularity levels [13]: (i) for coarse-grained pruned models, (ii) for fine-grained pruned models, (iii) for mixed-grained pruned models. Zhu et al. [13] presented a zero-skipping dataflow for feature. Although such method raised computing efficiency, zero elements in intermediate result still occupied storage resource. Lu et al. [14] proposed a weight-oriented dataflow for fine-grained pruned CNNs with little decoding cost. However, 2s-AGCN model differs from above convolutional workloads in that feature first goes through graph matrix multiplication. This weight-oriented design cannot skip corresponding graph computation. Li et al. [15] worked on PCONV pruning [16], a mixed-grained method. With weight-stationary dataflow designed on FPGA, Li et al. improved the computing efficiency by 14.7%-44%. However, this work still occupied storage space for huge scale of zero data like Lu et al., and its simple hardware structure could not tackle complex workloads in our task.

GCN Accelerators on FPGA. Many works on accelerating large graph's GCNs based on FPGA are presented in recent time. AWB-GCN [17] combined offline software averaging and runtime hardware workloads balancing on several large graph datasets. Zhang et al. [18] partitioned input data into smaller segments, then perform graph sparsification and node re-ordering for computation reduction and data locality. HyGCN [19] split GCNs workloads into *Aggregation and Combination* phases. Different hardware structures and dataflows were designed for two phases respectively. To sum up, above works focus on: (i) Leveraging and expanding graph adjacency matrixes sparsity, (ii) Avoiding irregularity and randomness of data distribution in graph computation, (iii) Keeping balanced workloads between PEs or computing phases, via offline and online ways. Unfortunately, graph in skeleton-based GCNs for action recognition models is dense and unchangeable. The data sparsity is embedded in temporal feature and pruned weights, not the graph. Moreover, action recognition GCNs behave not only like CNNs, but also like graph processing, leading to graph-specific design requirements. Therefore, current specialized architectures on CNNs and GCNs cannot efficiently accelerate target models since they just take one of the two sides.

IV. METHODOLOGY

This section introduces our hybrid pruning method for action recognition GCNs. The dataflow reorganization, coarse-grained and fine-grained pruning on temporal convolution are described respectively.

A. Dataflow Reorganization

After clipping self-similarity graph, the computing flow between graph and spatial convolution can be further summarized as (3), where G_k denotes $A_k + B_k$ from (2). The computing order is first high-dimensional matrix multiplication with G_k, then the spatial convolution of W_k and finally the result merging of three loops. In this dataflow, common pruning methods only functions in second phase but cannot optimize the graph computation, which occupies 49.83% of total workloads in (3).

$$f_{out} = \sum_{k}^{k_\nu} f_{in} G_k \bigotimes W_k \tag{3}$$

To better analyse the dataflow, we extract first two phases and its output *X*. A pixel can be described as $X(h, w, oc)$, where *h, w, oc* represent height, width and output-channel coordinates respectively. Then (4) can be deduced from (3) and *ic* is the acronym of input channel. Under the commutative law of multiplication, therefore (4) is transformed into (5). By reorganizing the computing order between graph phase and convolution phase, an opportunity for graph-skipping pruning is offered here. If the parameter element $W(1, 1, i, oc)$ is pruned to zero, the graph matrix multiplication in current output channel can be ignored. Further, if we set all convolutional parameters in *i* input channel as zero, then all graph computation can be skipped in current loops. The dataflow reorganization is then proposed when we apply above method to three loops in (3). Unlike conventional structure pruning method which drops different channels on filters, weights in specific input channels are all set as zero on every spatial filter in current convolutional blocks. In this way, not only the convolution workload is reduced, but also the graph computation is skipped.

$$X(h, w, oc) = \sum_{i=1}^{ic} (\sum_{p=1}^{25} f_{in}(h, p, i) \times G(p, w)) \times W(1, 1, i, oc) \tag{4}$$

$$X(h, w, oc) = \sum_{i=1}^{ic} (\sum_{p=1}^{25} G(p, w) \times f_{in}(h, p, i) \times W(1, 1, i, oc)) \tag{5}$$

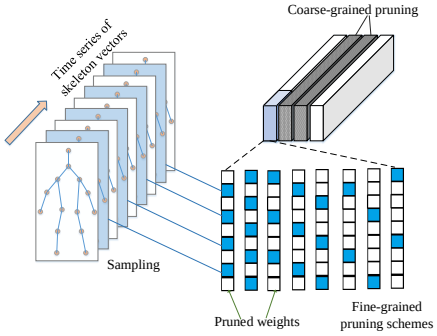

Fig. 3. The illustration of fine-grained pruning on temporal convolution. White elements are pruned while blue ones are kept. Every 9x1 kernel performs on time series of skeleton vectors, and the blue vectors are sampled by first pruning scheme.

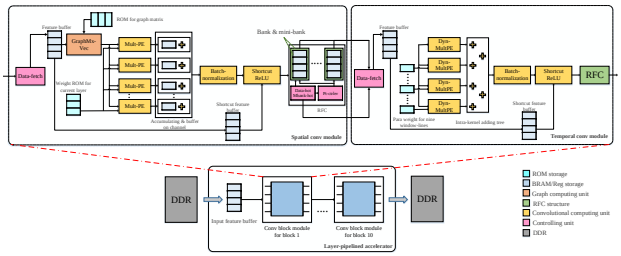

Fig. 4. The demonstration of overall architecture.

Since the graph-skipping strategy has been determined by dataflow reorganization, the next step is choosing the input channel to be pruned. Like other deep neural networks (DNN), features between convolutional layers are sparse and useful elements are unevenly distributed. Based on the observation that unstructured pruning method drops weight element with relatively small absolute value, we cut off the input channels which have least averaging absolute value. In this way, data reorganization prunes spatial convolutional weight and skips both graph and convolution computation.

B. Mixed-grained Pruning Method

In dataflow reorganization method, features in specific channels are not computed because such channels are pruned. The coarse-grained method then prunes corresponding temporal filters via connections in Fig. 2 with no extra accuracy loss. Moreover, this neighbour connection is hardware-friendly for that the number of pruned channels in spatial filters equals the number of pruned filters in temporal convolution. This inherent feature supports a balanced layer-pipelined architecture.

Coarse-grained pruning can provide 49.83%-88.96% compression ratio on temporal filters, depending on the pruning scheme in data organization phase. To further prune temporal convolutional weights, fine-grained pruning is proposed. The point of fine-grained method is that in temporal convolution, zero weight means not sampling current vectors in time order. Fig. 3 demonstrates details of sampling-like fine-grained pruning method. Several pruning schemes with different intervals and offsets are conducted on filters recurrently. By this means, the design of pruning scheme is turned into a sampling problem. We can simulate various sampling schemes on filters, with different sampling frequencies and phases represented by intervals and offsets. Experiments show that with proper pruning scheme, our fine-grained method can keep accuracy as well as discarding unimportant weight.

Conventional unstructured pruning methods randomly drop the weight elements with least absolute value, which are expensive and unbalanced on hardware. However, with determined cavity schemes, our fine-grained pruned model can be indicated by structured weight together with masks. Furthermore, we guarantee the balancing distribution of reserved weight by controlling start-points of different sampling patterns. Like Fig. 3 shows, in a loop of eight different pruning modes, weight elements in every position of kernels are evenly kept by two or three times. Compression ratio can be adjusted via fine-grained pruning design.

V. Architecture

This section introduces the detailed architecture of our accelerator, where all pruned convolutional blocks are mapped on chip.

Overview: Fig. 4 depicts the overall design of our layer-pipelined architecture. Conv block module constitutes the whole architecture, containing one spatial conv module (SCM) and one temporal conv module (TCM). To be more detailed, one conv block module processes on convolution block. Due that proposed pruning method reduces model size, all parameters and graph are stored in ROM storage on chip. Mult-PE is the basic computing unit of SCM, while Dyn-MultPE is the basic unit in TCM. Moreover, runtime sparse feature compression module (RFC) functions at the junctions of different layers to compact and store temporal results.

A. Spatial Conv Module

The main task of SCM is performing graph computation and pruned spatial convolution. Data-fetch controls the address of data-loading and decodes compact feature into sparse form, which is prepared for computing. Feature buffer receives and stores decoded data in order. The decoding process will be explained later. Sparse feature will first multiply with graph vectors, then conduct convolution with non-zero weight in Mult-PEs. Pruned channels are skipped and multiplication results are summed up in accumulating buffer on output channel. After batch-normalization operation, dataflow merges with original input activation, which is stored in shortcut feature buffer. ReLU function is combined with encoding function parts.

In order to combine graph computation and pruned convolution workloads, dataflow is organized as Fig. 5 shows. A line in feature buffer caches 25 data and the depth equals the number of kept channels in filter. When computing, buffer offers one line of original feature data. After computation with one column vector of graph, there generates one valid element $X(h, w, oc)$ in (4). Afterwards, feature buffer provides next cache line, which continues to produce $X(h, w, oc + 1)$.

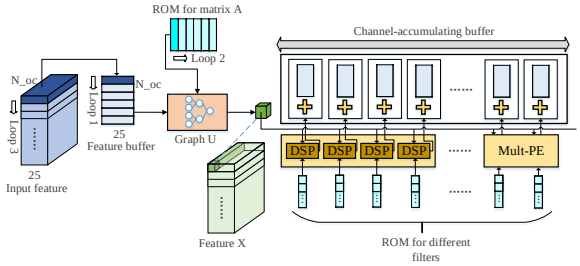

Fig. 5. Illustration of SCM dataflow organization.

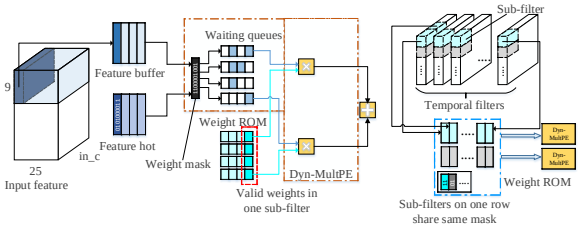

Fig. 6. Illustration of TCM's dataflow.

Following this mode, when all output elements on current output channel are computed, feature buffer returns to the first line and graph ROM switches to the vector in next column to prepare for $X(h, w+1, 0 : N_oc)$. When the workload of one row feature tensor is finished, feature buffer receives next row of tensor to start a new sub-loop. In this way, feature is produced in a channel-first order. Our dataflow reorganization method essentially abandons feature data on specific channels, so we skip corresponding workloads by not sending them to feature buffer.

Feature element is broadcast to all Mult-PEs. In the same channel-first order, weight ROM sends different filters' parameters into computing units. To match the pruned model, only non-zero weights are stored. Each Mult-PEs includes four DSPs, and by adjusting the number of Mult-PE, our design can fit into different layers. Results from parallel Mult-PEs are accumulated and buffered on channel direction as well. When the sum counter reaches the number of valid channels, current data will be transferred into post-processing modules.

B. Temporal Conv Module

TCM is designed to accelerate temporal convolution workloads, whose kernel size is 9×1. Fig. 6 shows the detailed information of TCM. Similarly, feature buffer stores decoded data from data-fetch. However, buffer width is turned from 25-data into 9-data, and the depth is tuned for holding an $9 \times 25 \times in\_c$ area of feature tensor. Additionally, feature's one-hot code is sent from data-fetch to feature hot storage as well. Valid weight together with its masks is stored on chip. As depicted in Fig. 3, several balanced fine-grained pruning schemes are conducted on leftover filters in recurrent ways, providing opportunities for structured weight storage. One temporal filter is divided into several $1 \times 1 \times 16$ sub-filters, thus, parameters can be folded into sub-filters format and then be stored in a recurrent mode. Moreover, Dyn-MultPEs are put across input channel and parallelizes on filter's rows. There are two reasons: (i) This parallel scheme can directly skip the abandoned filters in coarse-grained pruning, (ii) Each row of sub-filters is taken by one Dyn-MultPE and each function part handles one row of weight tensor. In this way, one Dyn-MultPE only needs to process weights derived from static cavity mask, such as four or six weights in Fig. 6. This design further eliminates data irregularity and scheduling uncontrollability.

Shown in Fig. 6, one row of a sub-filter is assigned to a Dyn-MultPE, which includes four or six waiting queues and several DSPs. Logic and operantion is performed first on weight mask and feature mask to skip the zero-feature and dropped weights. Then, valid feature enters one waiting queue, which is bonded to a non-zero weight in the sub-filter. To decrease the use of DSPs, dynamic data scheduling is designed by dispatching data from busy waiting lines to empty DSPs. Because multiplication in a Dyn-MultPE is part of the intra-filter computation, results need to be summed up and afterwards sent to the adder-tree. While dynamic data scheduling has advantages on saving DSP resource, it may increase the working delay at workloads-intensive cases. With the help of recurrent fine-grained pruning and statistic sparsity, we calculate the expectation of valid computation in one sub-filter and use it to guide the DSP occupation. The detailed method is illustrated in (6), where the number of kept weight in a sub-filter is assumed as six and s stands for feature sparsity.

$$E(D) = \sum d \times p(d) = 3(1-s)^3 + 3s^2(1-s) + 6s(1-s)^2 \quad (6)$$

C. Runtime Sparse Feature Compress

Despite layer-pipelined architecture poses great advantages on throughput, it has to store massive temporal computing results for shortcut task. RFC is presented to address this issue. Encoding, compact storage and decoding are included in RFC structure. Encoding process is combined with ReLU while decoding is embedded in data-fetch module. The whole structure of RFC is displayed in Fig. 7.

Encoding: At first, one feature vector is divided into several banks across channels. The width of each bank is 16 data-wise. ReLU function parts perform on banks, providing activation and one 16-bit hot code, which denotes the positive/zero value. Then valid elements are gathered at higher bits while unused bits are padded with zero. After that, mini-bank-hot code (mbhot) is generated according to the number of non-zero data in bank. Mbhot indicates which mini-banks are used in the bank storage. Encoding parts work in pipeline and during several working cycles, the whole vector is finally turned into compact format. Instead of compressing one vector as whole, we lower the encoding cost by setting bank as the finest grain of ReLU and encoding process.

Storage: The compact storage consist of bank storage units, which includes mini-banks, mini-bank-hot storage, data-hot storage and address controller pt. The key insight of bank

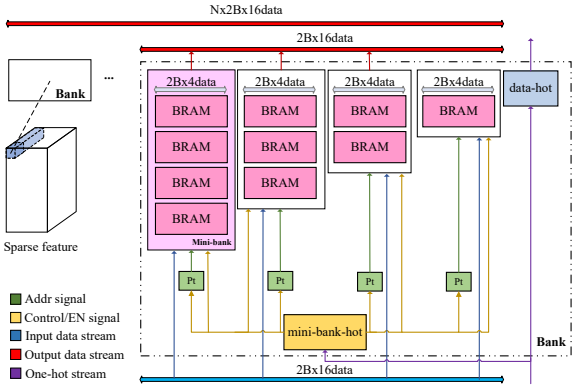

Fig. 7. Illustration of TCM's dataflow.

storage design is to keep access regularity on data-width dimension and reduce useless storage on data-depth dimension. When input data and hot codes are valid, mbhot sends enabling signal to every mini-banks and related pts. For example, if the input data-hot code is 0001_1100_0000_0111, meaning there are five non-zero data in the high bits of input data stream and the mbhot is 1100. The first mini-bank receives and stores four valid data, the second mini-bank keeps fifth valid data and three zero data. Their pt will self-add after this data-writing. Other mini-banks and their pt are not started. Similarly, when we need to load data from bank storage in order, mbhot enables related mini-banks and pts to output correct data. The output of disabled mini-banks is covered by zero. Without any random access schemes, compact data can be both stored and loaded in only one cycle.

Another issue on compact storage is to determine the volume of each mini-bank. Like deciding DSP's utilization, we can calculate the expectation of useful data based on offline sparsity. However, there always exists vectors drifting away from average sparsity. Denser vectors demand deeper mini-banks on the tail (the rightmost mini-bank in Fig. 7) while the lighter vectors merely occupy head mini-banks. In ideal cases, every vector is fit in bank-lines with no mini-banks unused and no vector truncated, but it is hard to precisely determine the number of valid data in every vector. Features sparsity distribution of each layer can help us to adjust the depth of mini-banks. For example, sparsity of feature is 50% and 25% of vectors has sparsity higher than 75%, 25%'s features are between 50%-75%, 25%'s data is between 25%-50% and 25% vectors' are below 25%. The mini-bank arrangement in Fig. 7 meets the demand of different density feature and reduces 37.50% storage resource compared with sparse format. In actual design, BRAM units have variable grains, which provides more flexibility.

Decoding: The decoding function is integrated in data-fetch module in SCM and TCM. Data-fetch not only controls loading address, but also translates compact data into sparse form. After receiving both data stream and data-hot codes from bank storages, parallel decoding modules perform on each banks output. Each translation part processes compressed feature in four pipeline stages, four data for one stage. Matched with encoding phase, the output of one bank is seen as the basic decoding grain, which further decreases the complexity of decoding circuit.

VI. Experiments

In this section, we evaluate our design on both software and hardware views. Our pruning method is explored on one V100 GPU using PyTorch, and accelerator architecture is implemented with Verilog HDL on Vivado 2018.3 IDE.

A. Validations on Hybrid Pruning Method

In our experiments, the proposed hybrid pruning method on 2s-AGCN model is compared with unstructured pruning. We also explore the impact of different pruning designs on accuracy, e.g., various fine-grained pruning schemes for temporal filters and channel-dropping modes in data reorganization phases.

Comparison: Fig. 8 illustrates the contrast between our hybrid pruning methods and conventional pruning means. Both unstructured pruning and hybrid pruning can elevate accuracy for deleting some unimportant weights and improving convergence performance. With same parameters reduction rate, our method achieves better accuracy performance in most cases. Additionally, we apply quantization on our pruned models. With negligible accuracy loss, float data is transformed into fix-point format, where eight bits are allocated to decimal part and eight to integer part. To further accelerate the proposed application-specific system, half of input skeleton vectors are skipped. Although input-skip method lowers prediction accuracy, it brings 50% reduction on total computation. Besides, the input-skip model with 86% compress ratio still keeps the accuracy no less than original model, so we choose this model as final accelerating target.

Exploration: Both data reorganization and fine-grained temporal pruning are fatal to model accuracy. To find the best pruning scheme, we conduct isolated experiments respectively. Additionally, based on our dataflow reorganization method, graph-skipping rate equals channel-dropping rate of this phase. Guided by feature sparsity in Fig. 9, we first set each layers channel pruning rate roughly equals its sparsity respectively. For higher parameter reduction ratio, we progressively raise compressing rate on layers and observe effect on accuracy,

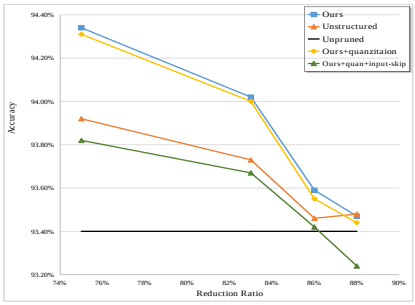

Fig. 8. Comparison results with unstructured pruning.

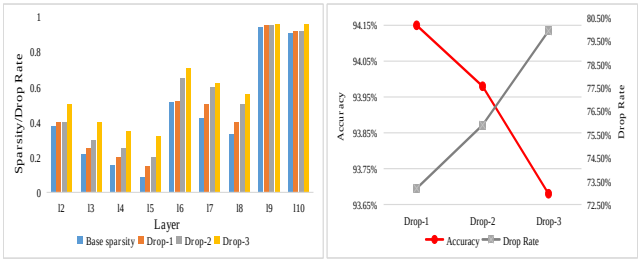

Fig. 9. Exploration on channel dropping.

as shown in Fig. 9. Drop-1, drop-2, and drop-3 stands for different channel pruning design. The spatial convolutional parameter in block 1 is not pruned for it only has three input channels. Also, mix-grained pruning on temporal convolution is excluded to validate data reorganization method. It reveals that with compress rate shifting away from base sparsity, model reduction is growing while accuracy is decreasing. We choose the Drop-1, which keeps the best accuracy, as our base dataflow reorganization design.

The fine-grained method is important in holding accuracy and keeping balanced-pruning, since coarse-grained means is totally decided by data reorganization. We carry out several experiments on fine-grained pruning, including different pruning intervals, offsets and pruning rates. All experiments are based on Drop-1 model in Fig. 9 and results are shown in Fig. 10. Pruning schemes in Fig. 10 are named as the combination of cav (cavity), pruning percent (50, 67 for instance) and intra-order. The size of rectangles below is 9×8, which denotes eight 9×1 kernels in loop. Cav-70-1 means the first cavity patterns with 70% reduction rate. With reduction rates expanding, model bears more accuracy loss in general. However, cavity patterns play an important role as well. With same compress ratio of 70%, cav-70-1 performs better than cav-70-2 on accuracy for more balanced weight pruning. Every weight line in cav-70-1 has two or three sampling chances, while in cav-70-2, different lines are kept from one time to four times. Balanced pruning schemes not only provide convenience for hardware, but also ensure the accuracy performance. The same situation happens between cav-75-1 and cav-75-2 as well. Taking both compress ratio and accuracy into consideration, cav-70-1 is chosen to be the final design.

B. Hardware implement

Dyn-MultPE: Dyn-MultPE works on the cav-70-1 cavity pattern, which means there are three Dyn-MultPEs needing to six waiting queues and six facing four waiting queues. Based on (6), different numbers of DSPs are settled in each layers Dyn-MultPEs. We also adjust the number of temporal convolutional PE to keep balance between pipeline stages. We choose some detail information to show in Table. II, where our dynamic data scheduling trades only 6.48% of longer delay for DSP reduction of 23.24%.

RFC: As stated above, RFC design relies on sparsity distribution. To optimize runtime compress storage, we refer to

TABLE II
UTILIZATION, WORKING EFFICIENCY AND MAX DELAY OF DYN-MULTPE.

layer	DSP in one PE	total DSP	efficiency	max delay
1	4/6	63	66.79%	0.00%
2	4/6	126	83.76%	3.70%
3	4/6	126	80.96%	0.00%
4	2/3	126	83.46%	7.40%
total		882	75.38%	6.48%
static		1149	57.86%	0.00%

TABLE III
FEATURE SPARSITY DISTRIBUTION OF SOME LAYERS

layer	I	II	III	IV
l1.sconv	<0.01%	29.35%	70.64%	<0.01%
l1.tconv	0.02%	94.73%	5.25%	0.00%
l2.sconv	0.00%	0.73%	75.79%	23.48%
l2.tconv	<0.01%	34.24%	65.76%	0.00%

offline sparsity distribution, which is partly shown in Table. III. Feature vectors are divided into four categories by their sparsity: 75%-100% (I), 50%-75% (II), 25%-50% (III) and 0%-25% (IV). According to our RFC design, vector of first category occupies one mini-bank, ones in II takes two, III takes three and IV takes four mini-banks. We can thus get the total BRAM blocks used for RFC structure. Comparison in Fig. 11 indicates that our RFC design brings 35.93% reduction on occupied BRAM blocks. Moreover, with almost same amount of used BRAM elements, RFC can finish data-loading in one cycle and encoding/decoding in four cycles, while CSC format usually needs 64 cycles to load data or decoding data serially. With less extra hardware cost and similar storage compress ratio, RFC structure achieves more regular data-access.

Overall performance: Our architecture is implemented on Xilinx XCKU-115 with frequency of 172MHz. The resource utilization is demonstrated in Table. IV, together with comparison with [10]. Experiments have proved that our design has superiority on peak performance, throughput and DSP

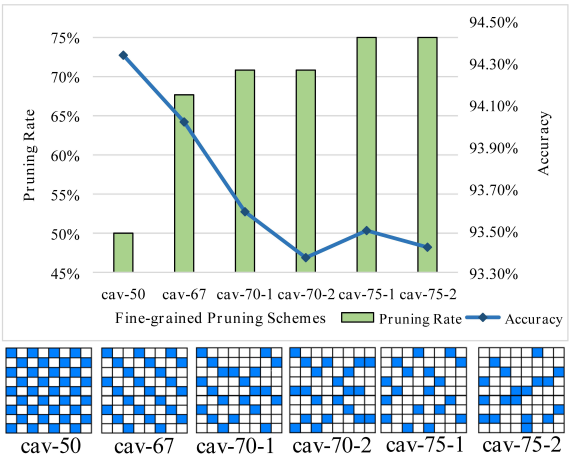

Fig. 10. Exploration on fine-grained pruning schemes.

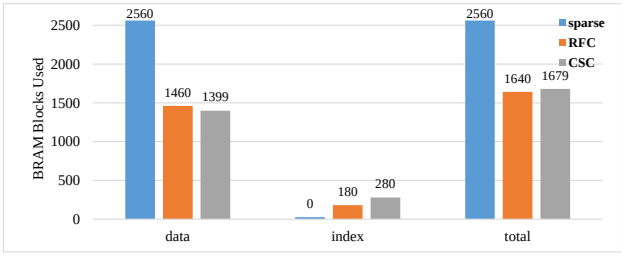

Fig. 11. Storage cost of three data formats.

efficiency. In Table. V, we compare the peak performance of ours and two GPUs. The original means testing program is the original version of 2s-AGCN, and the skip means input-skipping is applied. To fully use the memory in GPUs, target model runs with 200 or 700 samples in one batch on 2080Ti and V100, respectively. Although V100 has 14TFLOPS of peak performance on float data, our pruning methods provides 86% parameter reductions and 88% computation skipping efficiency. Moreover, our layer-pipelined structure does not need to exchange data with peripheral memory except for loading original input, which further increases performance. Compared with two main-stream GPUs, our accelerator provides 1.36x-9.19x of speedup, showing competitive performance.

VII. Conclusion

In this article, we propose a software-hardware co-design for action recognition GCNs: RFC-HyPGCN, including hybrid pruning method and a runtime sparse feature compress architecture with layer-pipeline. Firstly, a hybrid pruning method specific on action recognition GCNs is explored on 2s-AGCN. Secondly, we propose an architecture based on balanced pruned model. Finally, a runtime sparse feature compact format is designed to reduce zero-storage between layers. Experiments demonstrate that compared with conventional unstructured pruning, our method achieves better accuracy performance in most cases. The accelerator is implemented on Xilinx XCKU-115 FPGA. At the cost of negligible working delay, RFC reduces 35.93% of used BRAM and 23.24% of DSPs. Ours provides 22.62x speed-up and 59.41% elevation on DSP efficiency over another work on accelerating action recognition GCNs. On contrast to high-end GPUs, RFC-HyPGCN achieves 1.36x-9.47x speed-up on throughput.

References

[1] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Hierarchical representation learning in graph neural networks with node decimation pooling," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

TABLE IV
Utilization & performance comparison between ours and [10].

	dsp	bram blocks	LUT	dsp efficiency	peak perf	frequency	fps
ours	3544	1806	176776	0.322GOP/s/DSP	1142GOP/S	172Mhz	271.25
[10]	228	151	44457	0.202GOP/s/DSP	46GOP/S	188Mhz	11.99

TABLE V
Performance comparison between ours and high-end GPUs.

	ours	2080Ti-original	V100-original	2080Ti(w/o C)	V100(w/o C)	2080Ti-skip	V100-skip
throughput	271.25	29.53	69.38	45.42	98.87	104	199.09
speed-up		9.19	3.91	5.97	2.74	2.61	1.36

[2] K. Cheng, Y. Zhang, X. He, W. Chen, J. Cheng, and H. Lu, "Skeleton-based action recognition with shift graph convolutional network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[3] J. Gao, T. Zhang, and C. Xu, "I know the relationships: Zero-shot action recognition via two-stream graph convolutional networks and knowledge graphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 8303–8311, Jul. 2019.

[4] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[5] Y. Xiu, J. Li, H. Wang, Y. Fang, and C. Lu, "Pose Flow: Efficient online pose tracking," in *BMVC*, 2018.

[6] https://github.com/LiQiang0307/MobilePose pytorch

[7] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Hierarchical representation learning in graph neural networks with node decimation pooling," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2020.

[8] J. Li, T. Zhang, H. Tian, and S. Jin, "Sgcn: A graph sparsifier based on graph convolutional networks," in *Advances in Knowledge Discovery and Data Mining*, pp. 275–287, 2020.

[9] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Two-stream adaptive graph convolutional networks for skeleton-based action recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12026–12035, 2019.

[10] L. Ding, Z. Huang, and G. Chen, "An fpga implementation of gcn with sparse adjacency matrix," in *2019 IEEE 13th International Conference on ASIC (ASICON)*, pp. 1–4, IEEE, 2019.

[11] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "Ntu rgb+d: A large scale dataset for 3d human activity analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[12] A. Zisserman, J. Carreira, K. Simonyan, W. Kay, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, and M. Suleyman, "The kinetics human action video dataset," 2017.

[13] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 1953–1965, 2020.

[14] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–25, IEEE, 2019.

[15] N. Li, L. Liu, S. Wei, and S. Yin, "A high-performance inference accelerator exploiting patterned sparsity in cnns," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 243–243, IEEE, 2020.

[16] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5117–5124, 2020.

[17] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, *et al.*, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 922–936, IEEE, 2020.

[18] B. Zhang, H. Zeng, and V. Prasanna, "Hardware acceleration of large scale gcn inference," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 61–68, IEEE, 2020.

[19] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 15–29, IEEE, 2020.