VR-DANN: Real-Time Video Recognition via Decoder-Assisted Neural Network Acceleration

Zhuoran Song[1], Feiyang Wu[1], Xueyuan Liu[1], Jing Ke[1,2], Naifeng Jing[1,2]*, Xiaoyao Liang[1,2]*
[1] *Shanghai Jiao Tong University, China,* [2] *Biren Research, China*

***Abstract*—Nowadays, high-definition video object recognition (segmentation and detection) is not within the easy reach of a real-time task in a consumer SoC due to the limited on-chip computing power for neural network (NN) processing. Although many accelerators have been optimized heavily, they are still isolated from the intrinsic video compression expertise in a decoder. Given the fact that a great portion of frames can be dynamically reconstructed by a few key frames with high fidelity in a video, we envision that the recognition can also be reconstructed in a similar way so as to save a large amount of NN computing power. In this paper, we study the feasibility and efficiency of a novel decoder-assisted NN accelerator architecture for video recognition (VR-DANN) in a conventional SoC-styled design, which for the first time tightly couples the working principle of a video decoder with the NN accelerator to provide smooth high-definition video recognition experience. We leverage motion vectors, the simple tempo-spatial information already available in the decoding process to facilitate the recognition process, and propose a lightweight NN-based refinement scheme to suppress the non-pixel recognition noise. We also propose the corresponding microarchitecture design, which can be built upon any existing commercial IPs with minimal hardware overhead but significant speedup. Our experimental results show that the VR-DANN-parallel architecture achieves $2.9\times$ performance improvement with less than 1% accuracy loss compared with the state-of-the-art "FAVOS" scheme widely used for video recognition. Compared with optical flow assisted "DFF" scheme, it can achieve $2.2\times$ performance gain and 3% accuracy improvement. As to another "Euphrates" scheme, VR-DANN can achieve 40% performance gain and comparable accuracy.**

***Index Terms*—video decoder, motion vector, NN**

I. Introduction

Video recognition is a fundamental task in computer vision for video editing, surveillance and so on. With the wide adoption of machine learning in recent years, automatic video recognition has witnessed tremendous success with the help of convolutional neural networks (CNNs), particularly for video segmentation [3], [8], [47] and detection [1], [20], [46].

In nature, video is composed of a series of frames or images. With the success of deep learning on image recognition, one can intuitively transform the video recognition to a series of recognition on each individual frame. Given the fact that the majority of online video contents are usually encoded and compressed to save both storage and network bandwidth, the video recognition tasks on consumer devices have to first decode the video into normal frames, and then feed the DNN on a frame-by-frame process for feature recognition. The classic OSVOS [3] algorithm for video segmentation just works in this way. As illustrated in Fig. 1, OSVOS fully decodes the video bitstream, and then applies two large NNs, one for foreground and another for background on every frame. A modified FAVOS algorithm [8] utilizes the conventional tracking algorithm to localize object parts through the entire video, but still applies one large NN on every frame. Many works [13], [36], [41] have proposed multifarious models and optimization but only on the NN side. However the per-frame NN processing makes them incapable of real-time video recognition given the large amount of computation required.

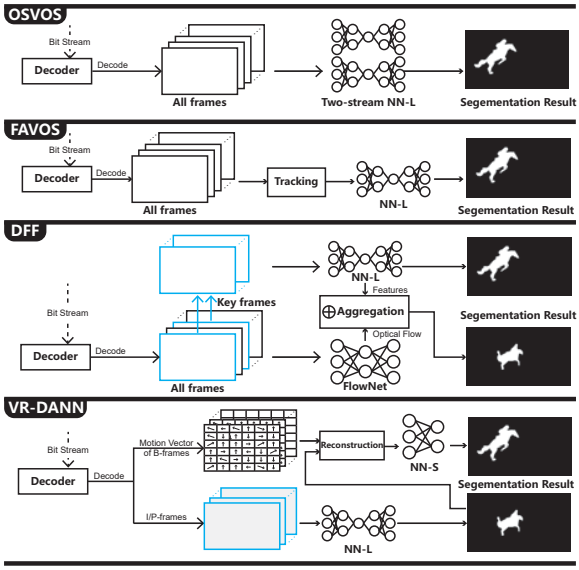

Fig. 1. Comparison of our VR-DANN to the state-of-the-art video recognition.

Aiming for faster video recognition, DFF [47] is trying to reduce NN workload by not fully processing every frame. As illustrated in Fig. 1, it classifies the frames into key and non-key frames. The key frames are segmented by a large NN that extracts features. For the non-key frames, DFF extracts the optical flow by FlowNet [11], and then aggregates the optical flow with the features from key frames to generate the segmentation results. Compared with OSVOS and FAVOS, DFF has better performance at the cost of accuracy as it simply classifies the key and non-key frames with a arbitrarily selected interval. As an alternative, differentiating the frames by temporal information in the video promotes state-of-the-art

We thank the reviewers for their thoughtful comments and suggestions. This work is partly supported by the National Natural Science Foundation of China (Grant No. 61772331, 61972242) and National Key Research and Development Program of China (Grant No. 2018YFA0701500). Naifeng Jing and Xiaoyao Liang are the corresponding authors.

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00063

video recognition accelerators [2], [5], [44], [48]. For example, Euphrates [48] simply moves the rectangle using the averaged motion vectors that are generated by ISP. EVA2 [2] first generates motion vectors by well-known motion estimation techniques, and then skips front layers by predicting the intermediate feature map using the motion vectors. However, they are inapplicable to video segmentation task and lack of exploration on expertise video decoder. In addition, the accelerators are still confined by tracking the temporal differences from the raw colored image representation with a fully decoded video, so the per-pixel based NN processing makes them hard for real-time video recognition.

In fact, the tempo-spatial information across images is the key idea for video compression, and has been well exploited in recent generations of video encoding standards such as H.264 and H.265 [33]. Somehow the observation has been long ignored in the state-of-the-art video recognition solutions. In this sense, we propose VR-DANN, a decoder-assisted NN acceleration for fast and accurate video recognition solution for the consumer SoC. We make the following contributions:

1) Modern video encoder classifies the video frames into I, P, and B frames. Being inspired that a B-frame can be completely reconstructed by I/P-frames, we envision that its segmentation results can also be reconstructed from the segmentation results (rather than raw images) from the I/P-frames. Because B-frames usually dominate a video stream, we can save a significant portion of NN computation for segmenting B-frames.
2) We propose the VR-DANN scheme that can leverage motion vectors for segmentation reconstruction on B-frames. Motion vectors are a small volume of tempo-spatial information inherently encoded in the video stream that capture the motion trajectory of an object. Different from any available schemes, our scheme does NOT require a fully decoded video. Instead, the decoder only needs to decode the I/P-frames, and output the inherent motion vector information in B-frames, so that our scheme will work as shown in Fig. 1. However, the reconstructed segmentation may lose details on the macro-block boundary, we propose a lightweight NN learned from the nearest I/P-frames to refine the segmentation results for B-frames.
3) We further propose a VR-DANN-parallel architecture to support the algorithm, which can be seamlessly integrated with any off-the-shelf NN accelerator and video decoder. It monitors the video decoding process for the required information (frame type, motion vector, block location) and links with the NN accelerator. This dedicated architecture can parallelize the reconstruction and synchronize multiple coupled dataflows between the decoder and the NN accelerator. It leverages the macro-block level independence within B-frames to optimize the global memory accesses. The decoder-accelerator co-design methodology jointly makes the proposed scheme accurate, practical and efficient.

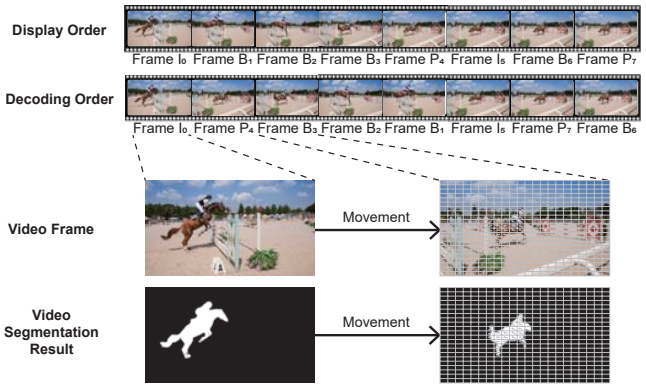

Fig. 2. Decoder ordering and a motion vector to reconstruct the segmentation.

The experimental results show that VR-DANN-parallel can achieve $5.7\times$ and $2.9\times$ performance gain and $4.3\times$ and $2.1\times$ energy reduction with less than 1% accuracy loss compared to "OSVOS" and "FAVOS". Compared to the optical flow assisted "DFF", we can achieve $2.2\times$ performance gain and $1.7\times$ energy reduction with even 3% accuracy improvement. Compared to the ISP assisted "Euphrates", we can achieve 40% performance gain with comparable accuracy.

To the best of our knowledge, this is the first attempt to closely link the video decoder and NN accelerator to form a unified SoC design for video recognition. By sharing the accurate tempo-spatial information between two application domains, we can save the decoder workload by just decoding a subset of video stream and save a large amount of computation and energy on the NN side by smartly leveraging the inherent motion vector information encoded in the compressed videos.

The remaining of this paper is organized as follows: Section II introduces the motivation. Section III describes the algorithm and Section IV elaborates the proposed architecture. Section V and VI present the experiment methodology and results. Section VII reviews the related work. Finally, Section VIII concludes this paper.

II. Video Decoding Preliminary and Motivation

We will first review the key concepts in modern video en/decoder. Taking the most up-to-date H.265 standard as an example in Fig. 2, for compression, it classifies a series of raw images into I, P and B frames and applies different strategies on a macro-block basis, typically with 8×8 pixels.

- For an I-frame, each macro-block undergoes an intra-frame encoding with a total of 14 prediction modes. For each mode, the encoder runs the predicting algorithm and calculates the sum of the absolute error (SAE) between the current macro-block and the macro-blocks already encoded within the current frame. The encoder will pick the mode and the macro-block with the least SAE.
- For a P-frame, the encoder searches in a wider range, among the macro-blocks already encoded in the current frame (intra-frame) and previous frames (inter-frame). The encoder applies a search strategy to search the macro-blocks in previous frames. For example, the encoder

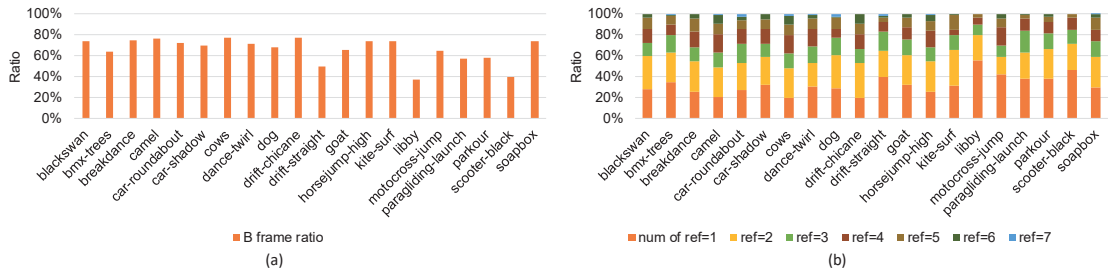

Fig. 3. (a) The ratio of B-frames in different videos; (b) The number of reference frames to reconstruct one B-frame across different videos.

searches the macro-block in the co-located position and upwards, left, down, and to the right macro-blocks. By comparing the SAE of these macro-blocks, the encoder can determine the macro-block.

- For a B-frame, the encoder searches in an even wider range, not only among macro-blocks in the previous frames but also the later frames. Similar with P-frames, the encoder also applies the search strategy on the macro-blocks and finds the macro-block with the least SAE.

The encoder denotes the frame and macro-block on which other blocks depends as the reference frame and macro-block. The encoder will insert a motion vector recording the referencing relationship into the bitstream to represent the motion trajectory of the macro-block. At the same time, the encoder records the decoding order of the frames according to the dependent relationship among frames and there can be forward and backward references for B-frames. For example, in Fig. 2, $(I_0, B_1, B_2, B_3, P_4, I_5, B_6, P_7)$ is the video display order while $(I_0, P_4, B_3, B_2, B_1, I_5, P_7, B_6)$ is the the actual decoding order, in which B_3 is dependent on I_0 and P_4.

On the decoder side, it decompresses the bitstream back to a series of normal frames according to the specified decoding order. For I-frames, a macro-block does an intra-frame prediction. For P/B-frames, the macro-blocks conduct both intra-frame and inter-frame prediction using the reference macro-block with the motion vector and the residual.

In this study, given the fact that any B-frame can be fully reconstructed by I/P-frames as its references, we envision that the object segmentation results of a B-frame can also be reconstructed by the segmentation results obtained from prior I/P reference frames. To achieve this goal, we should keep the reconstruction order identical to the frame decoding order recorded in the bitstream. This order guarantees that any B-frame will be processed after all its reference I/P-frames, implying that the reference segmentation results are already available. Fortunately, the reconstruction of B-frames can be easily done with the motion vectors that are have been encoded in the bitstream using a modern video decoder. As illustrated in Fig. 2, the example shows that if we can recognize a horseman on a horse in a reference I_0 frame, they can be roughly identified in the following B_3 frame simply through its motion vectors without running the full segmentation algorithm on B_3. Besides video segmentation, video detection can also work in the same manner so we will focus the discussion on segmentation in this paper.

TABLE I
NOTATIONS.

cur, ref	the current, reference frame index
S_{cur}, S_{ref}	segmentation results of current and reference frames
B_{cur}, B_{ref}	the current and reference blocks
$srcx, srcy$	the x and y coordinates of the macro-block in S_{ref}
$dstx, dsty$	the x and y coordinates of the macro-block in S_{cur}
NN-L	A conventional large DNN for video segmentation
NN-S	A much smaller NN for refinement

Since modern video encoders have already spent a lot of effort to search a wide range of reference frames for similarity matching around the neighboring n frames and macro-blocks with as much accuracy as possible, why shouldn't we make a better utilization of the inherent information during the video recognition process? This motivates us to propose a novel decoder-assisted NN accelerator design, which is different from all other available solutions striving for the pixel-level calculation in each of the raw image.

In addition, the number of B-frames often takes a large ratio compared to I/P-frames in a video. Fig. 3(a) reports the B-frame ratio of twenty different videos, which can reach 65% on average with the default encoder settings. The high ratio of B-frames accounts for the high compression rate in the modern encoding standard, which in turn implies that their segmentation results can be obtained without intensive NN computation via our proposed scheme. Fig. 3(b) statically summarizes the number of reference frames to decode one B-frame. We can see that the required number of reference frames for reconstruction or we call the reference "search interval" later in this paper, can be up to seven. In hardware, this phenomenon translates to random memory accessing and we therefore propose to reschedule the motion vectors for memory coalescing in Section IV-C to boost the performance.

III. The Proposed VR-DANN Algorithm

In this section, we will propose an algorithm to reconstruct and refine the segmentation in our VR-DANN scheme using notations listed in Table I.

A. The Algorithm

1) Reconstruction: As shown in Fig. 4, for either I-frame or P-frame, we will directly apply the conventional larger NN [8] (noted as "NN-L") to calculate the segmentation results. And for a B-frame, we will instead leverage the motion vectors pointing to its reference I/P frames for segmentation

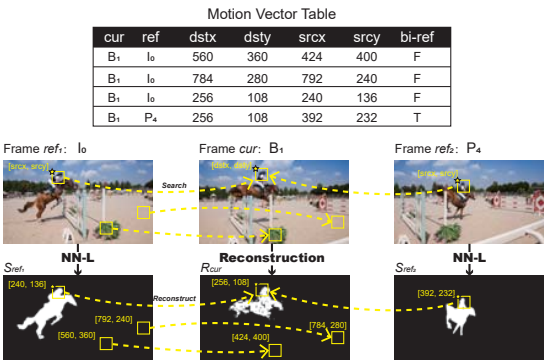

Fig. 4. The segmentation reconstruction process.

reconstruction. To reconstruct S_{cur} for the current frame B_1, we will first retrieve the segmentation results S_{ref} from the reference frames by locating the reference macro-block B_{ref} with a coordinate $(srcx, srcy)$ in the reference frame. The retrieved S_{ref} will become the intermediate reconstruction result for the current macro-block B_{cur} in a B-frame with the coordinate $(dstx, dsty)$. The $(srcx, srcy)$ of B_{ref} are pre-calculated during the video encoding process. The coordinates of $srcx$ and $srcy$ are quite random which may complicate the memory access pattern or the fine-grained memory control, as we will discuss in Section IV-C.

The example in Fig. 4 also shows the case that a B_{cur} in a B-frame macro-block with $(dstx, dsty)$ as $(256, 108)$ can have up to two motion vectors and need to be reconstructed with two macro-blocks, $(B_1, I_0, 256, 108, 240, 136, F)$ and $(B_1, P_4, 256, 108, 392, 232, T)$. We will use a bit, namely *bi-ref* which can also be extracted from the decoder to indicate such a case. So, we have to locate two reference macro-blocks with $(srcx, srcy)$ as $(240, 136)$ and $(392, 232)$ from two reference frames I_0 and P_4. Fortunately, the two motion vectors can be independently fetched from the motion compensation module in the video decoder. For this case, we can apply a simple mean filtering by averaging the segmentation results from both reference macro-blocks in a pixel-wise way. Note that the segmentation result for a pixel in a reference frame will be either 0 (black) or 1 (white) as shown in Fig. 4. This is different from the raw image data where each pixel contains 24 bits for colors. The pixel value after mean filtering can be 0 (black), 0.5 (gray), 1 (white) and is represented in 2 bits.

Note that the algorithm relies on the segmentation results of the reference frames, which is different from other techniques such as optical flow [9], [27], [46] that rely on every fully-decoded raw image. Meanwhile, our scheme is applied per macro-blocks basis, while other schemes are applied on pixel level, causing a significant amount of performance overhead. Finally, we use 1 or 2 bits for each pixel for B-frame reconstruction while other schemes use 24-bit pixels.

2) Refinement: Although the reconstruction using reference macro-blocks and motion vectors can be fast, it may suffer from noticeable noise impact in the reconstruction process. As seen in Fig. 4, the identified object bears unacceptable noise for real application into the state-of-the-art video segmentation. This is because the motion vectors work on the macro-block granularity. As a visual object often consists of multiple macro-blocks, the boundary of the object may become blurry.

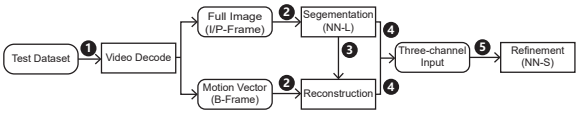

Fig. 5. The implementation flow of VR-DANN algorithm.

Given the fact that I/P-frames are segmented by NN-L, we further propose a small neural network (NN-S) to refine the object boundary by learning from the segmentation results of the reference frames. NN-S is 3-layer convolution neural network, including convolution, downsampling, convolution, upsampling, concatenate and convolution layers. To achieve this goal, we build sandwich-like three-channel images as the input to the NN-S, where the middle channel is the reconstruction results of current B-frame, and the first and third channels are the immediately preceding and following segmentation results of the reference I-frame and P-frame. We choose the immediately preceding and following reference frames because they are the temporally closest frames, so that the error can be minimized without strong propagation.

B. Applying VR-DANN Algorithm on Video Segmentation

To refine the reconstructed results of a B-frame, we will perform the inference using NN-S. As in Fig. 5, we first decode the input video for I/P-frames and get the motion vectors of the B-frames (denote as ❶). Then, we apply the NN-L inference using ROI SegNet described in FAVOS [8], which is the state-of-the-art model for video segmentation. This obtains S_{ref} of I/P-frames for future reference (denote as ❷). Next, we feed S_{ref} and the motion vectors of the B-frames for reconstruction (denote as ❸), which generates a coarse-grain boundary. Finally, we build the sandwich three-channel input (denote as ❹), and feed into NN-S for inference (denote as ❺) to refine the reconstruction and obtain the final segmentation results.

To train NN-S, we fully decode the training dataset for the I, P and B frames and obtain the motion vectors of B-frames. The I and P ground truth with the motion vectors in the B-frames are used to reconstruct the segmentation results, which are then combined with the ground truth I/P-frames as a sandwich image. Finally, we use the sandwich image as the input and the B ground truth as the label to NN-S for training the NN-S. As the structure of NN-S is much smaller than NN-L, the training overhead of NN-S is negligible compared with NN-L. Specifically, we only spend two epochs to train NN-S.

Our proposed VR-DANN can be extended to solve video detection problems using the same process. For video detection, we will first consider a rectangle box and the data inside as an object, while treating the other data as the background. Then we can apply the above reconstruction and refinement methods and obtain the detection results of the B-frames.

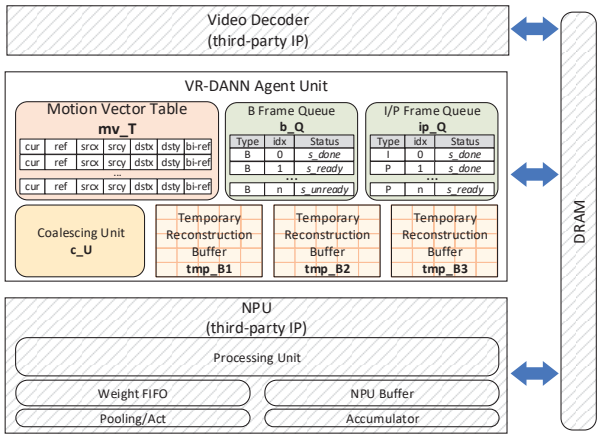

Fig. 6. The VR-DANN-parallel architecture overview.

C. Interaction with Video Encoder

The proposed VR-DANN algorithm may interact with the video encoder in the following ways:

Ratio of B-frames. The higher ratio of B-frames is, the less frequently NN-L is executed and the better the video segmentation performance is. But it also puts more challenges on the segmentation accuracy due to the less precise I and P segmentation results for referencing. The proportion of B-frames in modern video encoder can be configured and one can find the best performance with the satisfactory accuracy.

Searching interval of motion vectors. The motion vector is obtained by searching across n neighboring frames. Obviously, a larger searching interval benefits the accuracy because it is more likely to find a reference macro-block bearing more similarity to the current macro-block, which yields better reconstruction result. However, a larger searching interval complicates both the decoder hardware design and also stresses the memory bandwidth. We will propose a hardware-assisted technique in this paper to alleviate this problem.

Encoding standard. Our experiment finds that different encoding standards also affect the segmentation results. For example, compared to H.264, the latest H.265 standard adopts smaller macro-blocks. Larger macro-blocks can lead to poor reconstruction particularly on the boundary of the object, but can reduce the number of motion vectors in a B-frame and the computation workload. Generally, H.265 provides more accurate results and is more friendly to our scheme.

IV. The Proposed Architecture

In this section, we will propose the VR-DANN-parallel architecture design to efficiently enable the algorithm.

A. Architecture Overview

The proposed VR-DANN-parallel architecture is sketched in Fig. 6. We propose a standalone VR-DANN agent unit that can work with any off-the-shelf NPU (either a GPU or an ASIC) and video decoder. The agent unit consists of an I/P-frame queue (**ip_Q**), a B-frame queue (**b_Q**), a motion vector table (**mv_T**), several temporary reconstruction buffers (**tmp_B**) and associated control logic including coalescing unit (**c_U**).

Intuitively, the VR-DANN algorithm can work in a pure software implementation as in Fig. 5. However, the complete serial flow (denoted as VR-DANN-serial) will segment I, P and B frames according to their decoding order and this may hurt the performance. There are three major disadvantages: 1) because the segmentation of I/P and B-frames will exercise the NPU in a different way, the switching between NN-L and NN-S on NPU may hurt the performance by the frequent model reloading and kernel swaps. To be specific, the switching includes weight/input reloading, whose cost depends on NPU architecture(up to millisecond in GPGPU); 2) if not designed carefully, the reconstruction of B-frame and the processing on NPU will not be overlapped causing longer latency; 3) because the reconstruction process incurs lots of random memory accesses for the reference blocks, they are detrimental to the performance if not well scheduled.

The proposed VR-DANN-parallel architecture, particular the hardware-managed agent unit solves all the problems by introducing the independent **ip_Q** and **b_Q**, which can serve the NPU with two different types of NN workloads (NN-L and NN-S). The agent unit introduces several of **tmp_B** on-chip buffers to speed up the reconstruction of B-frames with well coalesced memory accesses, and the **mv_T** provides the motion vectors in B-frames pointing to the reference blocks in I/P-frames. These components are necessary and key to achieve a real-time video recognition with the cooperation from the traditional decoder and NPU.

B. Segmentation Ordering on NPU

As shown in Fig. 6, the agent unit is equipped with two asynchronous queues, an I/P-frame queue (**ip_Q**), and a B-frame queue (**b_Q**), for the reordering and executing of NN-L and NN-S on NPU according to the following rules:

1) The agent unit sets up the queues according to the frame information with their decoding order. The queues work like FIFO. Metadata of the I/P-frames are pushed into the **ip_Q** and that of the B-frames are pushed into the **b_Q**. The frame type can be extracted from the high-level parameter parser from the decoder [45], and their metadata such as the frame ID and address are stored in the queues after the decoder has written the raw image data into the global DRAM.

2) For an I/P-frame in **ip_Q**, it is ready for image segmentation. So its associated state will be set as *s_ready* once loaded into **ip_Q**. The **ip_Q** will instruct the NPU for NN-L processing of these images in the decoding order. The NPU can work on its own to handle the I/P images and models. Note that our scheme is NOT limited by the working principle of the NPU. Once the segmentation for an I/P image completes, the associated state is updated to *s_done* and the segmented results are written back to DRAM for later references.

3) For a B-frame in **b_Q**, its associated state is initially set as *s_unready* because it need to be reconstructed first. So it will first fill the **mv_T** with the motion vectors of the B-frame from the global memory. **mv_T** will then guide the **tmp_B** to

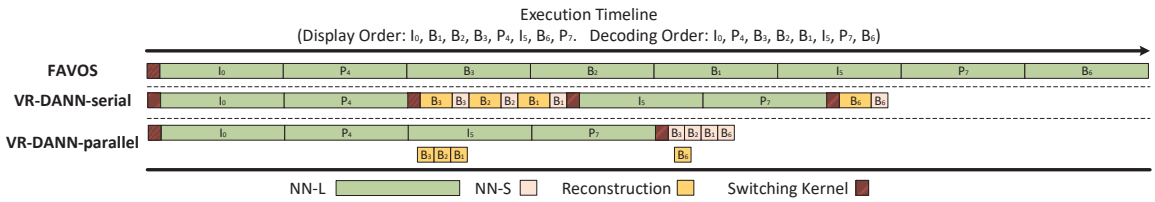

Fig. 7. The execution timeline of FAVOS, VR-DANN-serial and VR-DANN-parallel.

perform the reconstruction on a macro-block basis. Once the reconstruction is done, its associated state will be updated to *s_ready* for further refinement. The refinement requires a three layer NN-S, so we propose a coarse-grained time-multiplexing way to switch between NN-L and NN-S on the NPU. Once the refinement by NN-S has been completed, the associated state of the B-frame will be updated to *s_done* and the final segmentation results can be written back to DRAM.

4) The above process will be repeated until all the frames output from the decoder are segmented.

The agent unit may pick any ready frames from either **ip_Q** or **b_Q** for NPU processing. Because NN-L and NN-S can intervene with each other on frequent kernel and weight switching, the mode changing on NPU should be minimized. Since the two queues are asynchronous and independent, we propose a lagged queue switching to keep a stable NPU mode as long as possible under the prerequisite of not affecting the user experience. As the example in Fig. 7, assuming a video decoding order as $(I_0, P_4, B_3, B_2, B_1, I_5, P_7, B_6)$, traditional schemes like FAVOS require each I, P and B frame undergo a complete NN-L for segmentation, which is time-consuming and can hardly match the decoding speed. With the software solution VR-DANN-serial, we can dramatically reduce the NPU processing time since all the B-frames can be processed with a much smaller NN-S, at the cost of reconstruction time of B-frames. One issue here is the processing order on NPU strictly follows the decoding order so we will see a switching pattern of "NN-L, NN-S, NN-L, NN-S" as shown in Fig. 7. The frequent model and kernel switching, as well as the frame reconstruction insert many bubbles in the execution time line. With our lagged switching in VR-DANN-parallel, the two asynchronous queues will change the segmentation order to $(I_0, P_4, I_5, P_7, B_3, B_2, B_1, B_6)$, with only one switching as "NN-L, NN-S" on NPU as shown in Fig. 7. As we will see later, the time spent on B-frame reconstruction can also be completely hidden in VR-DANN-parallel.

In case that any queue is empty, we should switch to another queue for the full occupation of the NPU. Because the reconstruction of B-frames depends on the segmentation results of the leading I/P-frames, the scheme always initiates from the **ip_Q**. We always run a predefined number of I/P-frames from the **ip_Q**, after that we will switch to drain the **b_Q** to avoid the overstocking of B-frames. Since the reconstruction of B-frames depends on the leading I/P-frames, by comparing the frame ID of the tail of **b_Q** to the head of **ip_Q**, we can guarantee the correctness without any deadlock.

C. Motion Vector Rescheduling and Parallel Reconstruction

The VR-DANN architecture can avoid fetching the raw image data for B-frames saving a significant amount of memory bandwidth. However, the reconstruction of B-frames introduces new memory accesses such as motion vectors, part of the segmentation results from the reference frames, reconstructed B-frames and parameters for NN-S. The overall data volume from/to DRAM can be reduced because the segmentation frames has 1 or 2-bit value per pixel, while the full image has 24-bit colored value for each pixel. Importantly, the segmentation results in the reference frames may be loaded irregularly from DRAM because motion vectors can point to disperse locations in up to seven frames as investigated in Fig. 3(b), which may cause detrimental impact to performance.

In our study, given the fact that macro-blocks are independent to each other, we envision that the segmentation results can be independently reconstructed to maximize data reuse. We propose to reschedule the motion vectors in **mv_T** in favor of the reference block fetching.

We now explain the motion vector rescheduling by taking the example in Fig. 8. Suppose the **mv_T** has loaded eight motion vectors from three B frames (B_1, B_2, B_3), and they all point to some reference blocks in an I frame (I_0). We design a coalescing unit to generate memory requests with address index from ref and $srcy$ fields in the **mv_T**. In this way, we can coalesce multiple blocks in a single DRAM burst request. As shown in Fig. 8, the eight motion vectors are coalesced to four individual S_{ref} requests, i.e. $(ref, srcy) = (0, 11), (0, 13), (0, 2), (0, 5)$ and are forwarded to DRAM. Each request brings multiple block data and in turn all the eight reference block values. The coalescing unit can search into a wide range with multi-cycles operations. In our study, we find that a coalescing unit searching 32 motion vector entries simultaneously can well fulfill the requirement.

Once a reference block is returned, it will be dispatched to an on-chip **tmp_B** buffer according to the destination address $(dstx, dsty)$ recorded in the motion vectors in **mv_T** to finish the reconstruction of that macro-block. Note that since the source address $(srcx, srcy)$ address may not be aligned, data multiplexing should be applied for the right data before written into the **tmp_B**. For example in Fig. 8, when $(ref, srcy) = (0, 11)$ is returned, the motion vectors will fill **tmp_B1**, **tmp_B2** at destination $(8, 0)$ and $(24, 16)$, simultaneously and respectively. This out-of-order but parallel reconstruction, together with the motion vector coalescing, can greatly speed up the segmentation process of B-frames given

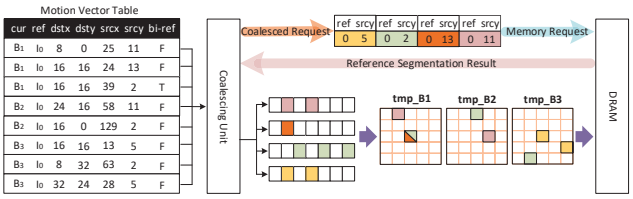

Fig. 8. Example on motion vector rescheduling for parallel reconstruction.

that we have multiple **tmp_B** buffers. Generally, we find that three **tmp_B** buffers should be enough to capture most of the inherent data locality and reconstruction parallelism because there are usually three B-frames sitting between an I-frame and P-frame in typical video streams. As illustrated in Fig. 7, with this proposed hardware-assisted reconstruction scheme (VR-DANN-parallel), we can hide the reconstruction latency of B-frames during NPU processing, and the entire execution time will be dominated by running NN-L and NN-S on NPU.

Once all the motion vectors belonging to the three B-frames are processed, the results in the **tmp_B** buffers will be written back to DRAM and the corresponding states will be set to *s_ready* meaning the three B-frames have been reconstructed.

D. Hardware Cost Analysis

Each pixel in **tmp_B** holds a 2-bit value, 00 for black, 01/10 for gray and 11 for white. For an 8×8 macro-block, the reconstructed segmentation results take 128 bits for on-chip storage. The reason we use two bits for a single pixel is because the macro-block may point to two reference blocks as shown in Fig. 4. Since each pixel in the segmented reference blocks takes only one bit (0 for black and 1 for white), the reconstructed block requires the mean filtering. The simplest way to conduct the mean filtering is to just combine the values read from both reference blocks. If both blocks read as 1 (white), the combined value becomes 11 which is white. Similarly, if both blocks read as 0 (black), the combined value becomes 00 which is black. If one block reads as 1 (white) and another reads as 0 (black), the combined value becomes 10 or 01, both cases stand for gray. The *bi-ref* field in **mv_T** can tell if the macro-block points to two reference blocks or not, and perform the mean filtering according. For a high-definition 854x480 video, each **tmp_B** buffer takes about 100KB, and we include three **tmp_B** buffers summing up to 300KB on-chip SRAM capacity as listed in Table II.

As shown in Fig. 6, besides the three buffers, the agent unit mainly introduces two queues and one table. The **ip_Q** has 8 entries. Each entry needs 8 bits for frame ID, 1 bit for frame type (I/P), 1 bit for frame status (*s_ready* and *s_done*) and 32 bits for frame address. Proportional to the typical B-frame ratio in a video, the **b_Q** has 24 entries. Each entry needs 8 bits for frame ID, 2 bits for frame status (*s_unready*, *s_ready* and *s_done*) and 32 bits for frame address. The **mv_T** contains 256 entries for macro-block-based reconstruction, which is sufficient for coalescing to hide the memory latency on hundreds of blocks in multiple **tmp_B** buffers. Each entry takes 1 bit to indicate if it references to two blocks or not, 4 bits for current indexing, 4 bits for reference indexing and 12 bits for each addressing field ($srcx$, $srcy$, $dstx$, $dsty$), which are wide enough for 4K videos. In summary, their total size will be less than 2KB as listed in Table II.

The reason why we are willing to pay the mild cost to design such a standalone agent unit is to hook up with any type of third-party NPU and decoder IPs. One may argue that the NPU can perform the reconstruction task as well but unless it is designed in a GPGPU style with strong programmability, this is not an easy task. Obviously we cannot assume any specific NPU design for the general application of our scheme. A software-based solution on CPU can also help but CPU is generally very inefficient for the large scale random memory accessing and parallel image reconstruction. This may also intervene with other upper level SoC tasks. If not handled properly, the latency of reconstruction may not be fully covered under the NPU processing time as illustrated in the VR-DANN-serial case in Fig. 7. In VR-DANN-parallel, with the lightweight agent unit, two conventionally separate application domains (AI and Video compression) can be seamlessly linked and unified to deliver better accuracy and greater performance.

V. Experimental Methodology

In the following sections, we will evaluate the accuracy and efficiency of VA-DANN, which is complementary to enable a real-time video recognition.

A. Validation on Algorithm Accuracy

Video segmentation. We use DAVIS [29], which is the most popular large-scale video dataset available in public. It consists of 50 high definition video sequences adding up to 4,219 and 2,023 frames for training and validation, respectively. The accuracy of video segmentation is usually measured in terms of F-Score and Intersection-over-Union (IoU). F-Score is defined as the weighted harmonic mean of the test precision and recall on a pixel level, while IoU measures the overlap rate of the segmentation result and the ground truth. We compare the segmentation accuracy of VR-DANN with the following state-of-the-art designs:

1) OSVOS [3], a classic method targeting video segmentation, which applies two deep CNNs on each frame. For comparison, we take the segmentation results directly reported from their paper.
2) DFF [47], which solves both segmentation and detection problems. It utilizes the FlowNet [11] on non-key frames and employs DNN on key frames. We reproduce their experiments using their open-source code for the results.
3) FAVOS [8], which segments objects in videos via tracking parts. We use their open-source code for the results.

We evaluate our VR-DANN algorithm on video segmentation by first extracting the motion vectors of each video using FFmpeg [34]. We run FAVOS to obtain the segmentation results of I/P frames which means we borrow the NN parameters used in FAVOS as the NN-L in our scheme. We then implement the reconstruction process referencing the segmentation of I/P-frames to obtain the reconstruction results

TABLE II
CONFIGURATIONS OF VR-DANN-PARALLEL ARCHITECTURE.

VR-DANN Agent Unit				NPU (Ascend 310)	
tmp_B	300KB	b_Q	126B	NPU Compute (INT8)	16TOPS
mv_T	1.8KB	ip_Q	42B	NPU Buffer	8MB
Agent Unit Frequency		600MHz		NPU Frequency	1GHz

for B-frames. Afterwards, we apply the Keras framework [17] to train NN-S. Finally, we execute NN-S inference to obtain the refined segmentation results for B-frames.

Video detection. We use Imagenet VID [31], which is a prevalent large-scale benchmark for video object detection. It consists of 3,862 videos from the training set and 555 videos from the validation set. The accuracy of video detection is usually measured in terms of the standard mean average precision (mAP) score. AP score is to take the average value of the precision across all recall values and mAP is the average of AP scores across all categories.

We compare the detection accuracy of VR-DANN with the following state-of-the-art designs:

1) SELSA [40], which aggregates features in the full-sequence level. For comparison, we directly take the detection results from their paper.
2) Euphrates [48], which solves detection tasks using image signal processor. Since Euphrates uses in-house datasets, we reproduce it on ImageNet-VID dataset.

The implementation of VR-DANN algorithm on video detection is similar to video segmentation.

B. Validation on Architecture Efficiency

We develop a cycle-accurate architecture simulator to model the VR-DANN architecture, especially the agent unit including the **ip_Q**, **b_Q**, **mv_T**, **tmp_B**, etc. We model the control logic to orchestrate the components, such as queue pop, push, the table entry access and state update with 1.67ns (600MHz) delay because they are all relatively light-weight hardware components. In addition, we integrate DRAMSim [37] into our simulator to model the external access to the global memory.

In our experiments, we apply the same hardware settings for NPU as in Ascend 310 [23], which is a commercial NPU widely-used in consumer electronic market. As illustrated in Table. II, the agent unit runs at 600MHz while the video decoder runs at 300MHz [45]. We build simple behavior timing models for the NPU and decoder, and integrate them with the agent unit. For the global memory, we use the default DDR3 in DRAMSim. In addition, we used CACTI [35] to estimate the energy and area of the on-chip queues and tables. For example, the 300KB 32-bank **tmp_B** in agent unit costs $2.0mm^2$ and $0.53nJ$ under TSMC-45nm technology.

We can dump the real motion vectors, the frame decoding order for reconstruction, and the activation and weights for NN-L and NN-S for NPU execution. For a fair comparison, we simulate the performance of OSVOS, FAVOS and DFF under the same NPU hardware as used in the VR-DANN architecture. To compare with a software only solution for VR-DANN algorithm, we also evaluate a serial software based implementation as discussed in Section III, termed as "VR-DANN-serial". The proposed VR-DANN hardware architecture is named as "VR-DANN-parallel".

VI. EXPERIMENTAL RESULTS

A. Video Recognition Accuracy

Fig. 9 compares VR-DANN to FAVOS for all the benchmarks in the suite for segmentation. From the plot, we can see that VR-DANN can match FAVOS accuracy on most of the videos. For "bmx-trees", "breakdance" and "motocross-jump", these videos mainly contain dramatic deformation on the visual objects, causing VR-DANN some difficulty in the refinement because the types of deformation vary across videos and the NN-S is hard to learn such features to a full degree. For "parkour" which has very fast moving objects inside, the motion vectors cannot precisely predict the exact location of the objects. To solve this problem, we can always refine the VR-DANN algorithm with fewer B-frame reconstruction while treating some B-frames as I/P-frames to pass through NN-L.

Fig. 10 reports the average F-Score and IoU results by applying OSVOS, DFF, FAVOS and VR-DANN algorithm on the DAVIS dataset using the default video bitstreams. From the plot, we can see that compared to OSVOS, VR-DANN improves the F-Score and IoU by 5.5% and 7.6% respectively. OSVOS is the original algorithm that passes the video through a large NN frame by frame. The accuracy purely relies on the learning capability of the network. However, VR-DANN can dig out the inherent information in the decoding process to assist the segmentation since the motion vectors carry important information as good hints for segmentation. Compared to DFF, VR-DANN improves the F-Score and IoU by 3.3% and 3.8%. DFF handles video frames differently at a predetermined fixed interval so that it might erroneously treat some important frames. Among the four algorithms, FAVOS yields the best accuracy because it first applies a conventional CV algorithm to track all the components in each frame and the tracking information will be used to assist the large segmentation NN so as to locate and identify finer-grain regions with higher accuracy. Compared to FAVOS, VR-DANN only lacks in the F-Score and IoU by about 0.4% and 0.5%, but can potentially gain significant performance as explained below. Note that there are several works [10], [32], [38] also make trade-off between accuracy and performance and they mentioned that less than 1% accuracy loss is tolerable.

For detection, Fig. 11 reports the average mAP under the ImagenetVID dataset. VR-DANN degrades the mAP by 0.8% on average compared to SELSA. We further classify the videos into fast, medium and slow groups according to the object moving speed in the videos. VR-DANN performs well for slow-moving videos and only degrades 0.5%. However, VR-DANN cannot handle fast-moving objects well, and may degrade the mAP by about 1.1%. In fact fast-moving objects could also be a problem for SELSA. Similar to segmentation, we can always reduce the ratio of B-frames to alleviate the problem for fast moving objects. To compare with Euphrates, we tune its key frame interval, i.e., Euphrates-2 in the figure

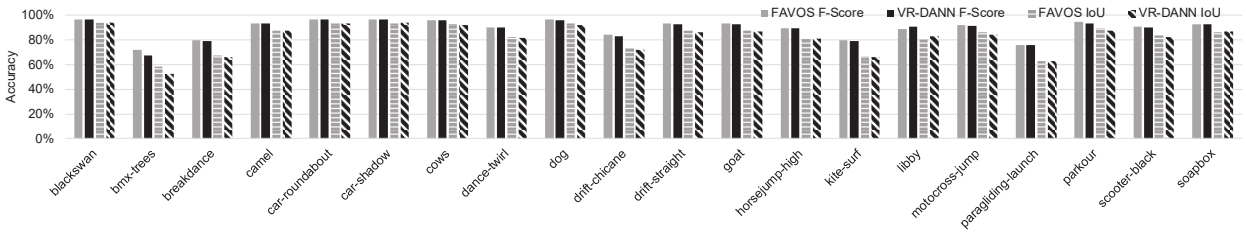

Fig. 9. Detailed segmentation accuracy of FAVOS and VR-DANN across different videos.

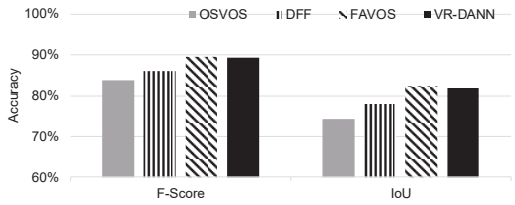

Fig. 10. Averaged segmentation accuracy results of different techniques.

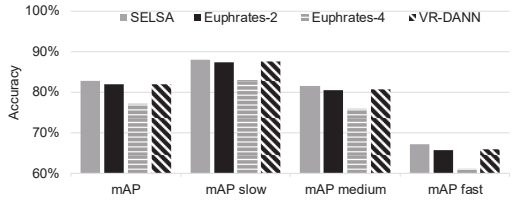

Fig. 11. Averaged detection accuracy of SELSA, Euphrates and VR-DANN.

denotes the key frame interval is 2. We find when Euphrates-2's accuracy(81.8%) is comparable to VR-DANN(81.9%), its performance is inferior by 40%. Euphrates-4's performance is enhanced but its accuracy sharply drops by 5.6%.

B. Performance and Energy Consumption

Fig. 12 provides the detailed execution cycles normalized to FAVOS across different videos. In the experiment, we use the default settings for the video encoder, so that the B-frame ratio is auto-tuned and the ratio may vary across videos. Since the benefit of our scheme has a close relationship with the B-frame ratio in the video, the performance of our scheme also varies across different videos. For example, VR-DANN-parallel improves the performance by 4.2× on "cows", but only 1.6× on "libby". Fig. 12 also shows the number of operations (TOPS) across different videos. For segmentation, the raw TOPS of a frame drops from 0.5TOPS (before) to 0.17TOPS (after optimization) on average. Moreover, since B-frames dominate, the TOPS can drop 60% on average.

Fig. 13 reports the performance and energy of all the studied algorithms by normalizing to FAVOS. For VR-DANN, we have two implementations. VR-DANN-serial is a pure software solution to serially implement the algorithm according to the decoding order. VR-DANN-parallel is the proposed parallel architecture with the aid of a specially-designed agent unit. In terms of performance, VR-DANN-serial improves the performance by an average of 3.9×, 2.0× and 1.5× compared to OSVOS, FAVOS and DFF. This is mainly because VR-DANN-serial can reduce a significant number of frames passing through the large NN for segmentation. It greatly simplifies the processing for B-frame segmentation with a much reduced three-layer small NN.

Fig. 13 also compares the proposed VR-DANN-parallel architecture with the software managed VR-DANN-serial. From the plot, VR-DANN-parallel can further improve the performance over VR-DANN-serial by 1.5× with the help from the standalone agent IP that can be seamlessly integrated into an existing SoC. As explained in Section IV-C, the ideal goal is to completely hide the latency of B-frame reconstruction and eliminate the kernel switching overhead on NPU. VR-DANN-parallel can approach nearly the ideal performance and from outside one may only observe the latency of kernels running on NPU with minimal switching cost. Our scheme can boost the real-time video recognition rate from 13fps to 40fps, matching the speed of the high-definition 854x480 decoder.

Energy wise, VR-DANN-parallel can reduce 4.3×, 2.1×, 1.7× and 1.1× consumption on average over the other four schemes. Compared with OSVOS and FAVOS, VR-DANN-serial can save a lot of computing and memory energy due to the much reduced workload on NN-S. VR-DANN-parallel can further save memory energy due to the dedicated memory coalescing scheme, and also reduce the NPU switching cost thanks to the asynchronous queuing scheme. VR-DANN-serial and VR-DANN-parallel exceed DFF because DFF spends lots of energy on searching the optical flow while VR-DANN succinctly uses video encoder for motion vectors, which is essential in modern video encoder. Note that the performance and energy gains are achieved without sacrificing accuracy.

Fig. 14 provides a break-down of the global memory accesses normalized to FAVOS to help understand the advantage of VR-DANN-parallel architecture. FAVOS consumes large DRAM bandwidth because of the large number of activation (every frame in the video with full color) and large number of weights forming NN-L. VR-DANN eliminates the full image retrieval from DRAM of all the B-frames by instead only fetching part of the segmentation results from the reference frames. The network parameters of NN-S are also greatly simplified compared with NN-L. Compared to VR-DANN-serial, VR-DANN-parallel further reduces the memory accesses by rescheduling the motion vectors through techniques like coalescing. The results in Fig. 14 justify these observations.

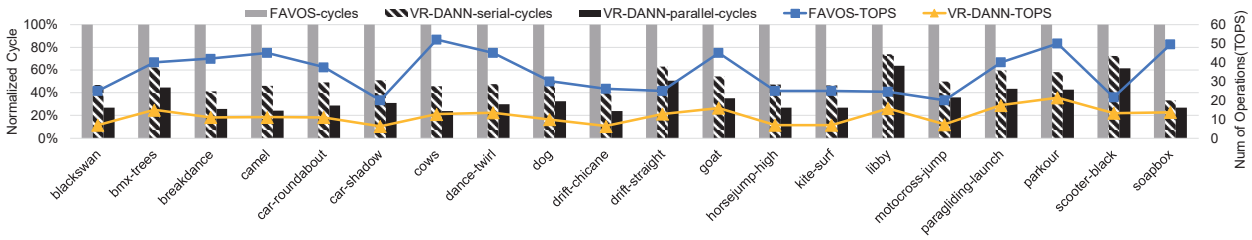

Fig. 12. Detailed execution cycles and number of operations of FAVOS, VR-DANN-serial and VR-DANN-parallel across different videos.

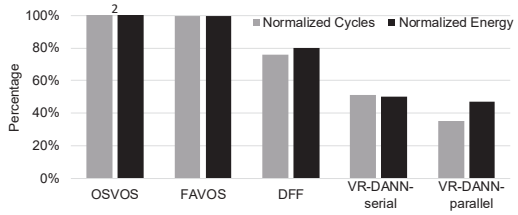

Fig. 13. Averaged performance and energy of different techniques on the segmentation task.

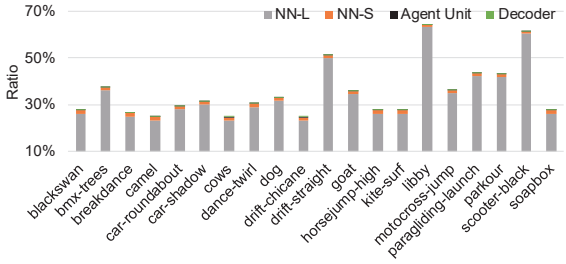

Fig. 14. DRAM access breakdown of VR-DANN-parallel.

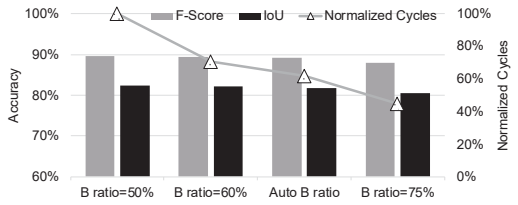

Fig. 15. The impact on segmentation accuracy and execution cycles of VR-DANN-parallel by varying the ratio of B-frames.

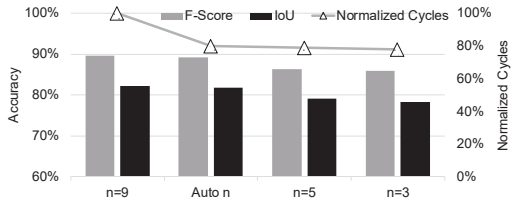

Fig. 16. The impact on segmentation accuracy and execution cycles of VR-DANN-parallel by tuning the search interval n.

C. Interaction with Different Encoder Settings

As discussed in Section III-C, there could be several factors that impact the segmentation accuracy and performance from the video encoding scheme.

Fig. 15 reports the segmentation accuracy and performance by varying the ratio of B-frames in a video stream. The label of "50% B ratio" represents that the B-frame ratio has been manually forced to 50% by setting the number (using "-b" command in FFmpeg) in the video encoder. The "auto B ratio" means that the B-frame ratio is auto-selected by the encoder itself. The auto-tuning typically results in around 65% B-frame ratio on average but a few videos may only have 37% to prioritize the video quality over the compression rate. From Fig. 15, we find that the F-Score and IoU suffer more accuracy drop at higher B-frame ratio. Too many B-frames may result in inaccurate motion vector predictions causing problem to the proposed scheme. However, too many B-frames are usually not acceptable because of the much lowered video quality. On the other hand, the performance actually improves with higher B-frame ratio. This is understandable because the more B-frames are, the more likely we can leverage the advantage of our scheme for speeding-up. VR-DANN-parallel generally works well with the auto-tuned encoder, yielding both satisfying accuracy and performance. Note that all the results reported before Fig. 15 use default "auto B ratio" option.

Fig. 16 studies the segmentation accuracy and performance by varying the reference frame search interval n. This parameter determines how many frames a B-frame can look ahead and after for the best matching for the macro-blocks. From the plot, we find that a larger n yields a higher accuracy for both F-Score and IoU, but at the cost of lower performance. Larger n means more reference frames can be searched but causes complicated accessing pattern to the global memory, reducing the overall efficiency of the hardware. Our motion vector scheduling technique can largely solve the problem and lineup most of the memory accesses. But with a too extreme $n = 9$, the efficiency may fall dramatically. Just like "auto B ratio", the encoder also provides "Auto n" as an option to set the search interval automatically. In all the experiments before Fig. 16, we chose "Auto n" as the default setting to balance the accuracy and performance.

Fig. 17 shows how different video encoding standards impact the segmentation accuracy. We find that H.265 generally delivers better accuracy than its prior generation H.264. This is because H.265 applies a more flexible motion prediction algorithm and a finer-grain macro-block size compared to H.264, at the cost of higher computing requirement on the more complicated encoder hardware. Since the modern encoder

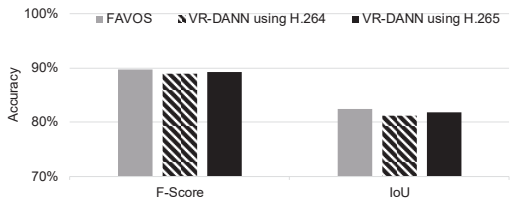

Fig. 17. The impact on segmentation accuracy by varying encoding standards.

has already paid the cost investing into more sophisticated prediction algorithms, it would be wise to maximally leverage the video-oriented information and develop corresponding machine learning schemes as we have done in this paper.

VII. Related Works

A. Video Object Segmentation and Detection

Deep CNNs have achieved great success on image recognition such as classification, detection and segmentation. So, they have extended from image to video domain.

1) Image Recognition: Deep learning is suitable for handling image recognition tasks. For object segmentation, fully convolutional networks (FCNs) [6], [25] have dominated the field. For object detection [14], [15], [19], [30], the regions with CNN features (R-CNN) series dominate. However, directly applying such image recognition models on all video frames costs unaffordable computation and energy.

Pruning and quantization techniques also work on image recognition tasks, and are complementary to our approach. To be specific, pruning is applied to make NN models to be sparse [18], [26]. Quantization is proposed to reduce the bitwidth of weights [21], [24].

2) Video Recognition: Given the image recognition limitations, researchers propose various NN models aiming at video recognition tasks. OSVOS [3] proposes the two-stream FCN model for foreground and contour branches, respectively. To achieve higher performance, FAVOS [8] proposes the part-based tracker to localize object parts. FAVOS then constructs the ROI SegNet, a robust but still large network to segment object. However, these mechanisms achieve high accuracy at the cost of high computation and energy.

It is acknowledged that image information varies slowly over video frames. Such data redundancy can be exploited to reduce the computation. To enable real-time video segmentation, several approaches are proposed to utilize the optical flow between frames [9], [27], [46] to accelerate the video recognition. Among these works, Zhu et al. [46] propose the deep feature flow (DFF), which is the first work to aggregate the optical flow with key features directly. The optical flow is extracted by the FlowNet [11], while the key features are obtained by convolving key frames with a large neural network. Moreover, Cheng et al. [9] propose Segflow, which simultaneously predicts pixel-wise object segmentation and optical flow in videos.

However, the key frames are determined by a fixed interval, which affects the accuracy. Moreover, the overhead of extracting the optical flow is also huge. Those drawbacks provoke us to leverage the information in video encoding.

B. Neural Network Accelerators

In recent years, various NN accelerators are proposed for image recognition tasks [4], [7], [12], [16], [22], [28], [43]. They are orthogonal to our proposal because they can accelerate the NN-L execution.

Compared with image recognition tasks, video recognition tasks drastically increase the computation. Some NN accelerators are proposed to capture the temporal information in video frames [2], [5], [39], [42], [44], [48]. For example, Zhu et al. [48] propose Euphrates that leverages ISP to accelerate detection task. Euphrates first requires ISP for motion vectors, which is not necessary in ISP. Afterward, Euphrates simply moves the rectangle using the averaged motion vectors. However, since video segmentation task is more complex with fine-grained boundary rather than a rectangle, so Euphrates is inapplicable to segmentation by only moving the rectangle. Moreover, Buckler et al. propose EVA2 [2], an software and hardware co-design. EVA2 speeds up video detection by running part of the network for each non-key frame. That is, it skips front layers by predicting an intermediate feature map and runs the following layers to the end. Due to down-scaling in pooling layers, the motion vectors will shrink or diminish if EVA2 skips too many front layers. So, EVA2 only applies to shallow networks, e.g. FasterM (5-layer) and Faster16 (16-layer) to balance the contradiction. But shallow networks cannot work well in the large dataset. Alternatively, TSVA [5] processes the computation only using the reduced bits on the difference of activations for each layer. Yuan et al. [44] propose a video processor that processes the key frame using the full precision. And the different data in each layer are divided into 4-bit and 8-bit blocks so that the efficiency of the processor can be enhanced.

The existing accelerators on video recognition take advantage of the redundancy of video frames. They either achieve high performance at the cost of accuracy [2], [39], or maintain the accuracy with limited performance enhancement [5], [42], [44]. Our proposed VR-DANN makes such a trial to closely link video decoder and NPU by preserving the accuracy and enabling higher performance.

VIII. Conclusion

Existing techniques for video recognition isolate the video decoding and NN schemes. Only raw image level information is passed to NPU for processing. In this paper, we propose the VR-DANN to rapidly reconstruct the recognition result of B-frames using the same philosophy (motion vectors) for video compression. We propose the VR-DANN-parallel architecture to efficiently implement the proposed algorithm via a well-designed and standalone agent unit to perform parallel reconstruction with memory coalescing in the consumer SoC. Our evaluation shows that the proposed scheme can outperform other existing schemes in performance, energy or accuracy.

REFERENCES

[1] Gedas Bertasius, Lorenzo Torresani, and Jianbo Shi. Object detection in video with spatiotemporal sampling networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 331–346, 2018.

[2] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. Eva^2: Exploiting temporal redundancy in live computer vision. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 533–546. IEEE, 2018.

[3] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 221–230, 2017.

[4] Srimat Chakradhar et al. A dynamically configurable coprocessor for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 38(3):247–257, 2010.

[5] Huixiang Chen, Mingcong Song, Jiechen Zhao, Yuting Dai, and Tao Li. 3d-based video recognition acceleration by leveraging temporal locality. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 79–90, 2019.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[7] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[8] Jingchun Cheng, Yi-Hsuan Tsai, Wei-Chih Hung, Shengjin Wang, and Ming-Hsuan Yang. Fast and accurate online video object segmentation via tracking parts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7415–7424, 2018.

[9] Jingchun Cheng, Yi-Hsuan Tsai, Shengjin Wang, and Ming-Hsuan Yang. Segflow: Joint learning for video object segmentation and optical flow. In *Proceedings of the IEEE international conference on computer vision*, pages 686–695, 2017.

[10] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al. Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 395–408, 2017.

[11] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[12] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[13] Raghudeep Gadde, Varun Jampani, and Peter V Gehler. Semantic video cnns through representation warping. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4453–4462, 2017.

[14] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[16] Vinayak Gokhale et al. A 240 g-ops/s mobile coprocessor for deep neural networks. In *CVPR Workshops*, pages 682–687, 2014.

[17] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[18] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[20] Congrui Hetang, Hongwei Qin, Shaohui Liu, and Junjie Yan. Impression network for video object detection. *arXiv preprint arXiv:1712.05896*, 2017.

[21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[22] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[23] Heng Liao, Jiajin Tu, Jing Xia, and Xiping Zhou. Davinci: A scalable architecture for neural network computing. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–44. IEEE, 2019.

[24] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[26] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017.

[27] David Nilsson and Cristian Sminchisescu. Semantic video segmentation by gated recurrent flow propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6819–6828, 2018.

[28] Maurice Peemen et al. Memory-centric accelerator design for convolutional neural networks. In *ICCD*, volume 2013, pages 13–19, 2013.

[29] Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 724–732, 2016.

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[32] Zhuoran Song, Bangqi Fu, Feiyang Wu, Zhaoming Jiang, Li Jiang, Naifeng Jing, and Xiaoyao Liang. Drq: Dynamic region-based quantization for deep neural network acceleration.

[33] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[34] FFmpeg team. Ffmpeg, 2019.

[35] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. Cacti 5.1. Technical report, Technical Report HPL-2008-20, HP Labs, 2008.

[36] Carles Ventura, Miriam Bellver, Andreu Girbau, Amaia Salvador, Ferran Marques, and Xavier Giro-i Nieto. Rvos: End-to-end recurrent network for video object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5277–5286, 2019.

[37] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. Dramsim: a memory system simulator. *ACM SIGARCH Computer Architecture News*, 33(4):100–107, 2005.

[38] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[39] Paul N Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas Kolala Venkataramanaiah, Jae-sun Seo, and Matthew Mattina. Fixynn: Efficient hardware for mobile computer vision via transfer learning. *arXiv preprint arXiv:1902.11128*, 2019.

[40] Haiping Wu, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Sequence level semantics aggregation for video object detection. In

Proceedings of the IEEE International Conference on Computer Vision, pages 9217–9225, 2019.

[41] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[42] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 129–144, 2018.

[43] Hoi-Jun Yoo et al. A 1.93 tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big data applications. In *2015 ISSCC*, pages 80–81. IEEE, 2015.

[44] Zhe Yuan, Yixiong Yang, Jinshan Yue, Ruoyang Liu, Xiaoyu Feng, Zhiting Lin, Xiulong Wu, Xueqing Li, Huazhong Yang, and Yongpan Liu. A 65nm 24.7j/frame 12.3mw activation-similarity- aware convolutional neural network video processor using hybrid precision, inter-frame data reuse and mixed-bit-width difference-frame data codec. In *2020 International Solid-State Circuits Conference*, 2020.

[45] Dajiang Zhou, Shihao Wang, Heming Sun, Jianbin Zhou, Jiayi Zhu, Yijin Zhao, Jinjia Zhou, Shuping Zhang, Shinji Kimura, Takeshi Yoshimura, et al. 14.7 a 4gpixel/s 8/10b h. 265/hevc video decoder chip for 8k ultra hd applications. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 266–268. IEEE, 2016.

[46] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 408–417, 2017.

[47] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2017.

[48] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. *arXiv preprint arXiv:1803.11232*, 2018.