A Computationally Efficient Neural Video Compression Accelerator Based on a Sparse CNN-Transformer Hybrid Network

Siyu Zhang[1], Wendong Mao[2], Huihong Shi[1], and Zhongfeng Wang[1,2], *Fellow, IEEE*

[1] School of Electronic Science and Engineering, Nanjing University, P. R. China
[2] School of Integrated Circuits, Sun Yat-sen University, P. R. China
Email: syzhang@smail.nju.edu.cn, maowd@mail.sysu.edu.cn, shihh@smail.nju.edu.cn, zfwang@nju.edu.cn

***Abstract*—Video compression is widely used in digital television, surveillance systems, and virtual reality. Real-time video decoding is crucial in practical scenarios. Recently, neural video compression (NVC) combines traditional coding with deep learning, achieving impressive compression efficiency. Nevertheless, the NVC models involve high computational costs and complex memory access patterns, challenging real-time hardware implementations. To relieve this burden, we propose an algorithm and hardware co-design framework named NVCA for video decoding on resource-limited devices. Firstly, a CNN-Transformer hybrid network is developed to improve compression performance by capturing multi-scale non-local features. In addition, we propose a fast algorithm-based sparse strategy that leverages the dual advantages of pruning and fast algorithms, sufficiently reducing computational complexity while maintaining video compression efficiency. Secondly, a reconfigurable sparse computing core is designed to flexibly support sparse convolutions and deconvolutions based on the fast algorithm-based sparse strategy. Furthermore, a novel heterogeneous layer chaining dataflow is incorporated to reduce off-chip memory traffic stemming from extensive inter-frame motion and residual information. Thirdly, the overall architecture of NVCA is designed and synthesized in TSMC 28nm CMOS technology. Extensive experiments demonstrate that our design provides superior coding quality and up to 22.7× decoding speed improvements over other video compression designs. Meanwhile, our design achieves up to 2.2× improvements in energy efficiency compared to prior accelerators.**

***Index Terms*—Neural video compression, CNN-Transformer, fast algorithm, pruning, hardware accelerator**

I. Introduction

Video data is rapidly growing due to the increasing demand for applications like virtual reality and surveillance. Compressing video is crucial to reducing transmission and storage costs under a limited bandwidth budget, and real-time video decoding in practical scenarios is highly desirable. During the past decades, several video coding standards, including H.264/AVC [1], H.265/HEVC [2], and H.266/VVC [3], have been established. However, conventional hand-crafted methods struggle to deal with complex textures and rapid motions and are less versatile for various scenarios and demands.

This work was supported in part by the National Key R&D Program of China under Grant 2022YFB4400604. *(Corresponding author: Wendong Mao, Zhongfeng Wang.)*

Recent years have witnessed the flourishing of brand-new NVC models [4]–[9], which integrate deep neural networks (DNNs) into hybrid video coding frameworks. Wherein the inter-frame compression, which employs the previous frames to predict the subsequent ones, has been the mainstream. Lu *et al.* [4] first employed two auto-encoder-style networks for NVC. Subsequently, numerous well-designed variants, including methods based on residual coding [5]–[7], conditional coding [8], and other technologies [9], have been proposed to further enhance coding efficiency. Compared to plain models [10] oriented to classification or object detection, the NVC models pose new challenges for real-time deployment: 1) NVC models generally comprise multiple independent modules, such as motion compensation, feature extraction, and residual compression, causing irregular data dependencies and extensive memory access. 2) Various operations, including deconvolutions (DeConvs) and deformable convolutions (DfConvs), are incorporated into NVC models, resulting in serious ***memory conflicts and computational imbalances***. 3) NVC models are sensitive to low-bit quantization and high sparsity, bringing challenges to the lightweight designs of model compression. 4) Practical applications of NVC require ***real-time processing of high-definition (HD) video streams***, causing substantial transmission burdens. All in all, the stringent requirements for memory and processing speed underscore the significance of devising lightweight model optimizations and efficient solutions for deploying NVC models.

Despite limited researches [11], [12] achieved deployments of NVC, they did not carry out dedicated designs and optimizations for specific hardware platforms. Moreover, many researchers focused on developing specialized accelerators [13]–[16] for certain operations within DNN models. However, directly applying them to NVC presents serious problems: 1) Existing accelerators struggle to handle complex data dependencies and memory access within NVC models. 2) Most accelerators mainly optimize dataflow and resource configurations for partial operations, making it difficult to simultaneously support the diverse operations within NVC models. 3) Extensive off-chip interactions for intermediate features such as motion and residual information incur sig-

979-8-3503-4859-0/DATE24/© 2024 EDAA

nificant power consumption and transmission burdens, which are unbearable for conventional accelerators. 4) Although there are currently accelerators individually supporting pruned fast convolutions (Convs) [15] or DeConvs [16], the differences in the patch size and computational patterns between these two operations pose challenges in achieving simultaneous support for both. Therefore, these problems necessitate in-depth research to enhance the performance and deployability of NVC on resource-limited devices.

HD video data in RGB or YUV format is typically stored on cloud servers as encoded bitstreams. These bitstreams are then distributed to users via the Internet. To recover visualized video streams, *users perform* ***real-time video decoding*** *on their mobile devices repeatedly* [7], [11], using previously decoded frames and recently received bitstreams. In light of this, we propose an efficient accelerator for NVC, named NVCA. The main contributions of this paper are listed as follows:

- At the algorithmic level, we present the CTVC-Net to improve compression quality by capturing multi-scale correlations. Additionally, we propose a fast algorithm-based sparse strategy by combining the pruning and fast algorithms in a dual approach, reducing computational complexity while maintaining compression efficiency.
- At the hardware level, we develop the reconfigurable sparse computing core to support sparse Convs and DeConvs based on the fast algorithm-based sparse strategy. We also incorporate a novel heterogeneous layer chaining dataflow to reduce the off-chip interactions caused by extensive inter-frame motion and residual features.
- The overall architecture of NVCA is meticulously developed and synthesized utilizing TSMC 28nm CMOS technology. Experimental results show that our design not only delivers superior compression performance and decoding speed over other designs but also outperforms other hardware designs in terms of energy efficiency.

II. Background

NVC models follow the traditional hybrid coding frameworks. Motivated by [5], all components operate within the ***feature space***, effectively reducing spatial-temporal redundancy and facilitating accurate motion estimation and compensation. As depicted in Fig. 1, the process begins with the transformation of the current frame X_t and the reference frame $\hat{X}_{t-1}$ from the pixel domain to the feature domain via a **feature extraction** module. These transformed features are denoted as F_t and F_{t-1}. To analyze the inter-frame motion relationships, a **motion estimation** module learns motion vectors O_t to compress redundant temporal information between F_t and F_{t-1}. Subsequently, O_t undergoes lossy compression through an auto-encoder-style network in a **motion compression** module, resulting in the reconstructed motion $\hat{O}_t$. Then $\hat{O}_t$ is used to compensate for F_{t-1} and generate the predicted feature $\bar{F}_t$ by a **deformable compensation** module. Following this, the spatial prediction error R_t between $\bar{F}_t$ and F_t is compressed via a **residual compression** module. The encoded motion and residual information is quantized and formed into bitstreams for transmission. Finally, by combining the reconstructed residual feature $\hat{R}_t$ with $\bar{F}_t$, a **feature reconstruction** module transforms the results $\hat{F}_t$ from the feature domain back to the pixel domain. This process results in $\hat{X}_t$, which is then stored within the decoded frame buffer. It is evident that multiple independent modules introduce complex data dependencies and extensive memory access, thereby posing significant challenges for the deployment of NVC models.

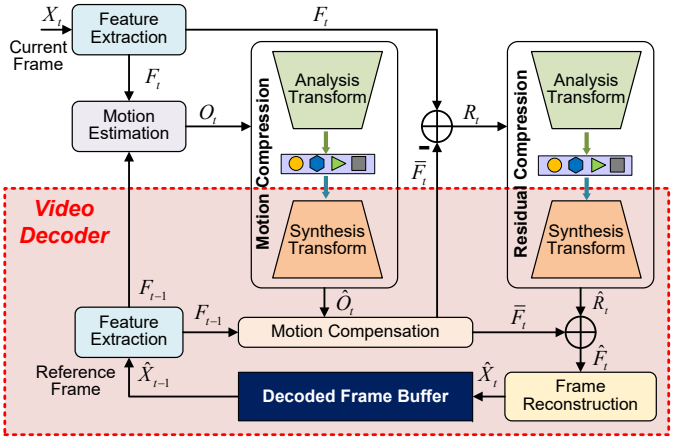


Fig. 1. Visualization of the entire process for the end-to-end NVC models. The decoding part of the NVC models is denoted by a red dashed box.

III. Proposed Novel NVC Model

A. CNN-Transformer Hybrid Video Compression Network

1) Network Structure of CTVC-Net: Our CTVC-Net follows the end-to-end NVC models in Fig. 1. Wherein the corresponding topologies and associated hyper-parameter settings for each module are provided in Fig. 2 (a)-(e).

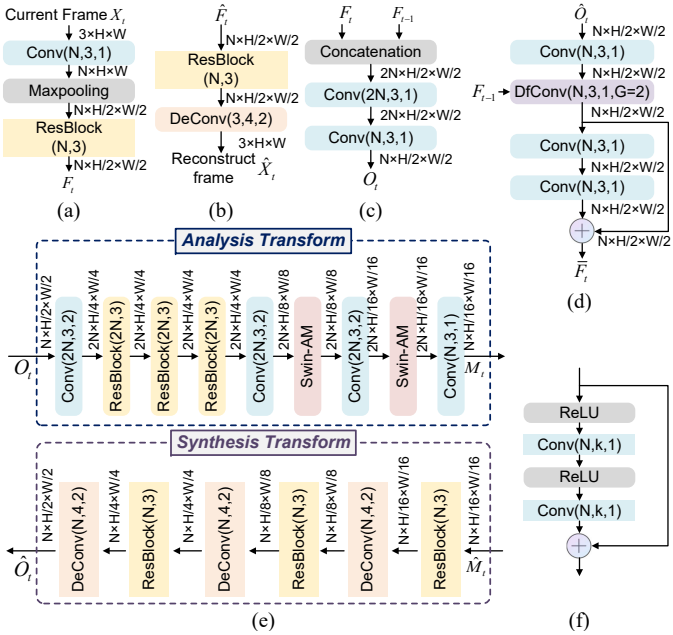


Fig. 2. Network structure of CTVC-Net. (a) Feature extraction. (b) Feature reconstruction. (c) Motion estimation. (d) Deformable motion compensation. (e) Motion (residual) compression. (f) Residual block (ResBlock). "Conv(N, k, s)" and "DeConv(N, k, s)" present the output channel number, kernel size, and stride of the Conv and DeConv, respectively. G of "DfConv(N, k, s, G)" means the group number for the DfConv.

2) Swin-Transformer-Based Attention Module (Swin-AM): Considering the strong correlations between spatially neighboring pixels, global semantic information is unsuitable for video compression [17]. Thus, we propose Swin-AM to enhance the compression efficiency of CTVC-Net. With the Shift window-based self-Attention (SwinAtten) [18], Swin-AM confines the global receptive field to local non-overlapped windows while integrating the *multi-scale structure information*, as depicted in Fig. 3. Specifically, Swin-AM comprises three branches. **Branch 3** manages the residual connection. **Branch 2** utilizes stacked ResBlocks to generate intermediate features. **Branch 1** employs SwinAtten and stacked Convs to mine multi-scale spatial correlations within local windows, producing window-based spatial-channel attention masks. In addition, to address the absence of feature connections across different windows, we design consecutive Swin-AMs with varying shift values Shf in CTVC-Net to bridge *cross-window connections*. In this way, Swin-AM effectively guides adaptive bit allocations and enhances compression efficiency by capturing local and global correlations.

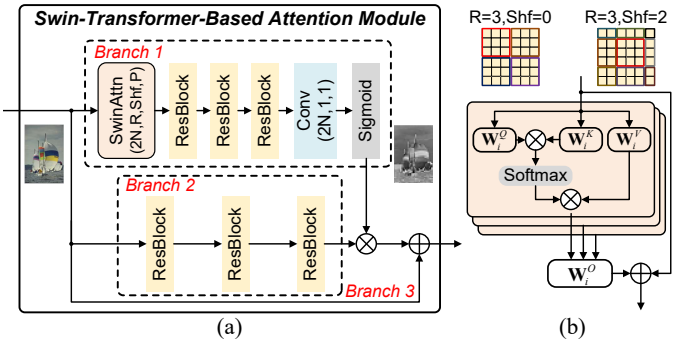


Fig. 3. Description of Swin-AM. (a) Network structure. $(2N, R, Shf, P)$ expresses the output channel number, window size, shift value, and head number. (b) Details of SwinAtten.

B. Fast Conv and DeConv-Based Sparse Strategy

Considering NVC models' sensitivity to quantization and pruning, we seek to reduce the computational complexity of CTVC-Net by *a dual approach* that combines pruning and fast algorithms. In the decoder of CTVC-Net, Convs and DeConvs play crucial roles in its high complexity. Although there are dedicated accelerators individually support pruned fast Convs or DeConvs, the intrinsic distinction between them hinders concurrent support for both. Thus, we present a fast algorithm-based united sparse strategy to reduce algorithm complexity.

1) Fast Conv and DeConv: Inspired by [19] and [13], we can express the fast Conv and DeConv by *a common formula*:

$$\mathbf{V} = \mathbf{A}^{\mathbf{T}}\left[\left(\mathbf{G}\mathbf{W}\mathbf{G}^{\mathbf{T}}\right) \odot \left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A}, \tag{1}$$

where $\mathbf{X}$, $\mathbf{W}$, and $\mathbf{V}$ denote the input patch sized at $p \times p$, the weight sized at $k \times k$ and the output patch sized at $m \times m$, respectively. $\odot$ signifies the Hadamard product, and $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{G}$ are transform matrices. Compared to individually computing each element within the output feature map, fast algorithms leverage structural similarities to generate output patches. This approach significantly reduces the computational complexity associated with both Convs and DeConvs.

To express the fast Conv and DeConv, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{G}$ can be defined by different hyper-parameters. Specifically, for the ***fast Conv*** (Winograd algorithm) [19] $F(m \times m, k \times k)$, the dimensions of $\mathbf{X}$ are $p \times p = (m+k-1) \times (m+k-1)$, and multiplication number is $\mu \times \mu = (m+k-1) \times (m+k-1)$. For example, the transform matrices of $F(2 \times 2, 3 \times 3)$:

$$\mathbf{A}^{\mathbf{T}}_{Conv} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}. \tag{2}$$

It means that given a 4×4 input patch, a 3×3 Conv producing a 2×2 output patch requires 16 multiplications, whereas a standard Conv needs 36 multiplications. Regarding the ***fast DeConv*** (FTA) [13] $T_r(m \times m, k \times k)$ (*please **note** that it is the DeConv-tailored method and not based on Winograd algorithm*), the dimensions of $\mathbf{X}$ are $p \times p = \lceil (k + r \times s - 1)/s \rceil \times \lceil (k + r \times s - 1)/s \rceil$, where $m = r \times s$, r represents the order. In this case, $\mu \times \mu = (k + (r-1) \times s) \times (k + (r-1) \times s)$. For example, for $T_3(6 \times 6, 4 \times 4)$ with a stride of $s = 2$, the transform matrices are as follows:

$$\mathbf{B}^{\mathbf{T}}_{DeConv} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix}, \mathbf{G}_{DeConv} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \tag{3}$$

$$\mathbf{A}^{\mathbf{T}}_{DeConv} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{4}$$

2) Transform-Domain Weight Pruning: According to Eq. (1), each element of $\mathbf{E} = \mathbf{G}\mathbf{W}\mathbf{G}^{\mathbf{T}}$ is derived by a weighted sum of uncertain elements in $\mathbf{W}$. Similarly, each element of $\mathbf{V}$ can be obtained via a weighted sum of uncertain weights $\mathbf{E}$ and inputs $\mathbf{X}$. Since the contributions of each element $\mathbf{E}_{i,j}$ to $\mathbf{V}_{c,d}$ are distinct, directly pruning $\mathbf{E}_{i,j}$ based on the magnitudes ignores their actual importance. Thus, we employ an *importance factor matrix* $\mathbf{Q}$ to scale the magnitude of $\mathbf{E}_{i,j}$:

$$\mathbf{Q}_{i,j} = \sqrt{\sum_{\substack{0 \leqslant c,d \leqslant m-1, \\ 0 \leqslant q,v \leqslant p-1}} \mathbf{H}^2_{c,d,i,j,q,v}} \quad 0 \leqslant i,j \leqslant \mu - 1, \tag{5}$$

$$\mathbf{H}_{c,d,i,j,q,v} = \mathbf{A}_{i,c} \cdot \mathbf{A}_{j,d} \cdot \mathbf{B}_{q,i} \cdot \mathbf{B}_{v,j}. \tag{6}$$

Then, a mask matrix $\mathbf{M}$ is generated based on a predefined sparsity ρ to indicate the redundant transform-domain weights:

$$\mathbf{M}_{i,j} = \begin{cases} 0 & \mathbf{Q}^2_{i,j} \cdot \mathbf{E}^2_{i,j} < \zeta \\ 1 & \mathbf{Q}^2_{i,j} \cdot \mathbf{E}^2_{i,j} \geqslant \zeta \end{cases} \quad 0 \leqslant i,j \leqslant \mu - 1, \tag{7}$$

where ζ denotes the pruning threshold. Subsequently, the sparse Conv and DeConv can be both formulated as follows:

$$\mathbf{V} = \mathbf{A}^{\mathbf{T}}\left[\left(\mathbf{M} \odot \mathbf{G}\mathbf{W}\mathbf{G}^{\mathbf{T}}\right) \odot \left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A}. \tag{8}$$

IV. Neural Video Compression Accelerator

A. Overall Hardware Architecture

In Fig. 4, the NVCA involves a Sparse Fast Transform Core (SFTC), a Deformable Convolution Core (DCC) and a global controller. The DCC, designed to support DfConvs, follows our previous work [14] and is briefly discussed here. The SFTC flexibly executes fast Convs or fast DeConvs by a switch flag. The computational logic involves the **Pre-processing Unit array (PreU array)**, the **Post-processing Unit array (PostU array)**, and the **united Sparse Computing Unit array (SCU array)**. The former two perform the domain transform, while the SCU array handles Hadamard products of specific data accessed by indices. On-chip memory includes the **Weight Buffer**, **Index Buffer**, **Input Buffer**, and **Output Buffer**. The Weight and Index Buffer store non-zero weights and their corresponding indices. The Input Buffer stores inputs and inter-layer features based on a novel *heterogeneous layer chaining dataflow*. DeConvs' outputs are written to the Output Buffer and then transferred to the **External Memory**. Besides, **FIFO** and the **Reshuffle Network** reorder temporary input or output data before sending it to the subsequent logic.

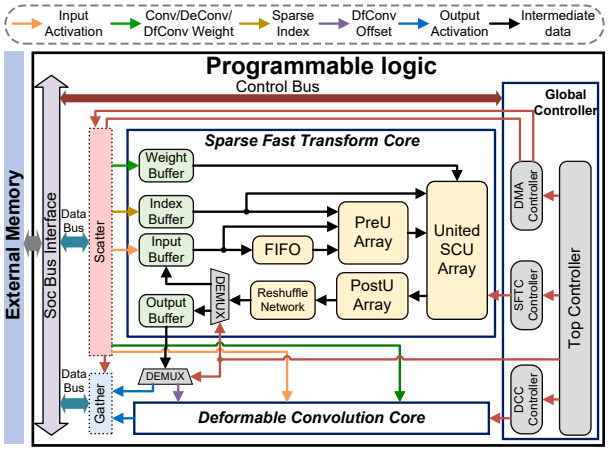


Fig. 4. Top-level block diagram of the overall architecture.

B. Sparse Fast Transform Core

Using the proposed fast algorithm-based sparse strategy to flexibly support sparse Convs and DeConvs with minimal hardware overhead, we apply $F(2\times2, 3\times3)$ for 3×3 Conv, which carry out 16 multiplications and $T_3(6\times6, 4\times4)$ for 4×4 DeConv which involves 64 multiplications.

1) Reconfigurable Computing Units: The domain transform units of SFTC are depicted in Fig. 5. The PreU, including 32 1D-PreUs, is responsible for mapping input patches $\mathbf{X}$ from the spatial domain to the transform domain $\mathbf{Y} = \mathbf{B^T X B}$. Similarly, the PostU, including 24 1D-PostUs, performs the inverse transform $\mathbf{V} = \mathbf{A^T U A}$, where $\mathbf{U}$ and $\mathbf{V}$ represent the input and output patches of the PostUs.

To promote data reuse in both spatial and temporal dimensions, we develop the united SCU array that flexibly supports ***fine-grained structured sparsity*** while maximizing parallelism. As shown in Fig. 6 (a), Pif PreUs and Pof PostUs are organized as a PreU array and a PostU array,

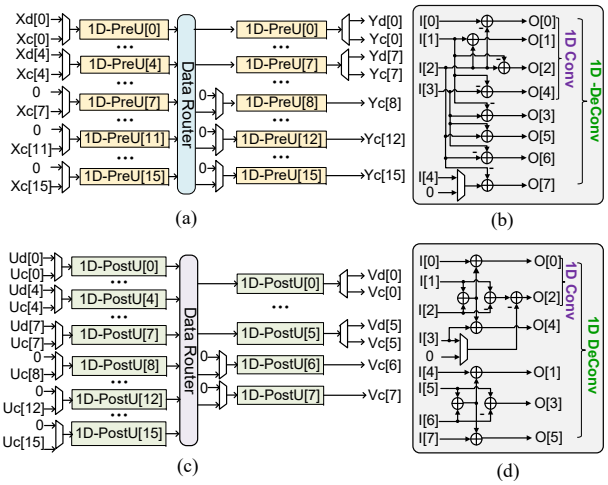


Fig. 5. Illustration of domain transform units. (a) The Pre-pocessing Unit (PreU). (b) 1D-PreU, which is a part of (a). (c) The Post-pocessing Unit (PostU). (d) 1D-PostU, which is a part of (c).

respectively. The unrolling of Pif input channels and Pof output channels occurs along the row and column directions, forming a united SCU array of $Pif \times Pof$ SCUs, as presented in Fig. 6 (b). Each SCU incorporates 64ρ multipliers for processing either one sparse DeConv or four sparse Convs. It selects specific inputs based on indices by a non-zero element selector, enabling element-wise multiplications with the compressed weights. After finishing the parallel Hadamard products, we reduce all input channels in the transform domain and only send the final sums to the PostU array for executing the inverse transform.

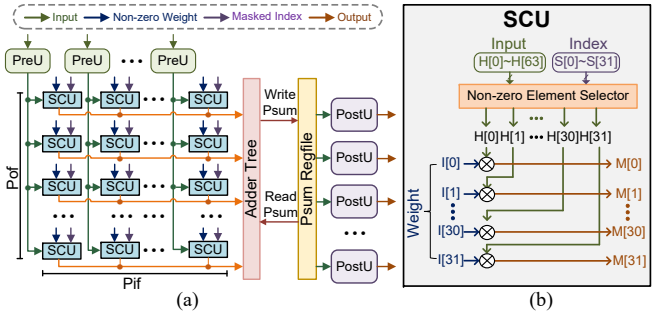


Fig. 6. Computing array for sparse Convs and DeConvs. (a) The united SCU array. (b) Details of SCU, which is a part of (a).

2) Heterogeneous Layer Chaining Dataflow: Compressing HD videos typically involves processing and transmitting extensive motion and residual information. Additionally, multiple independent modules lead to complex dataflow and memory access patterns. All these problems incur considerable energy consumption and limit throughput due to constrained interface bandwidth. To address this issue, we propose a novel heterogeneous layer chaining dataflow to reduce massive off-chip interactions. In contrast to existing research [10] of merging stacked Convs through layer fusion, the proposed method can achieve the fusion of both Convs and DeConvs while simultaneously supporting pruning and fast algorithms.

CTVC-Net decoder comprises numerous serial structures composed of two Convs followed by a DeConv, which are considered as a **chain**. In Fig. 7 (a), generating the output feature map D with 6 rows requires the input feature map A

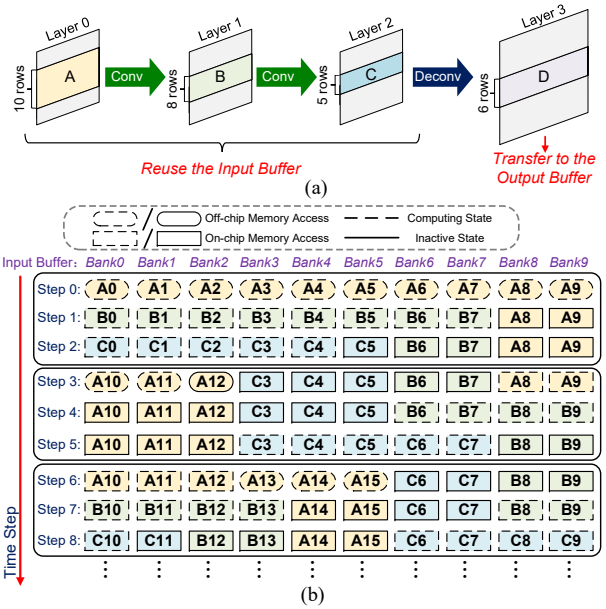


Fig. 7. The heterogeneous layer chaining dataflow. Please note that the feature maps in (a) correspond directly to those in (b) in terms of color. (a) The heterogeneous layer chain is composed of Convs and DeConvs. (b) Runtime scheduling of the Input Buffer.

with 10 rows, the intermediate feature map B with 8 rows, and the intermediate feature map C with 5 rows. To execute a ***complete*** fast DeConv, the Input Buffer includes 10 banks, denoted as Bank 0, Bank 1, and so forth. Since Conv preserves feature resolution, inter-layer features can reuse the Input Buffer through reasonable runtime scheduling. As presented in Fig. 7 (b), where Ai means the i-th row of the feature map A is sent to $i\%10$-th bank. Each bank operates in either a computing or inactive state, determined by the row number required for fast Convs and DeConvs. Once some banks' data has participated in the calculations and will never be used again, these banks will be overwritten with new data from other layers. By this means, intermediate results can be stored on chip and leveraged as inputs for adjacent layers, economizing massive off-chip interactions.

V. Evaluation

A. Experiment Setup

We employ the Vimeo-90K dataset [20] for training, and the HEVC Class B [2], UVG [21], and MCL-JCV [22] datasets for inference. Peak signal-to-noise ratio (PSNR) and multi-scale structural similarity index (MS-SSIM) [23] are utilized to quantitatively assess video quality, and bit per pixel (bpp) measures bit allocations for motion and residual. BDBR(%) [5] indicates the average percentage of bit rate savings at the same PSNR (MS-SSIM) compared with others. In our experiments, we set hyper-parameters like $N = 36$, $Pif = Pof = 12$, and maintain a consistent sparsity level of $\rho = 50\%$. We quantize floating-point (FP) weights and activations into fixed-point (FXP) format with 16 and 12 bits, respectively. To assess the effectiveness of hardware design, a cycle-accurate simulator [24] is developed for reliable performance estimation. Additionally, we verify the simulator against RTL implementation to ensure correctness. We synthesize unit energy and area using Synopsys Design Compiler (DC) with the TSMC CMOS 28*nm* HPC+ technology library.

B. Algorithm Evaluation

Fig. 8 illustrates the rate-distortion (RD) curves of our design on two benchmark datasets, showcasing our superior performance compared to other methods [1], [2], [4]–[6], [8]. Our design achieves **the lowest bit consumption at the same compression quality**. Besides, in contrast to conventional hybrid video coding standards [1], [2], we consistently deliver the best results across all datasets. Table I summarizes the BDBR (%) comparisons for PSNR and MS-SSIM. We find that under 50% sparsity, our design achieves 35.19% and 51.30% bit rate savings over the H.265 standard in terms of the PSNR and MS-SSIM on the UVG dataset. This reflects that by combining the Transformer-based model, our CTVC-Net effectively extracts multi-scale correlations to improve compressed video quality while significantly reducing bit rates. Besides, **the sparse CTVC-Net maintains excellent video compression efficiency** compared to the dense version.

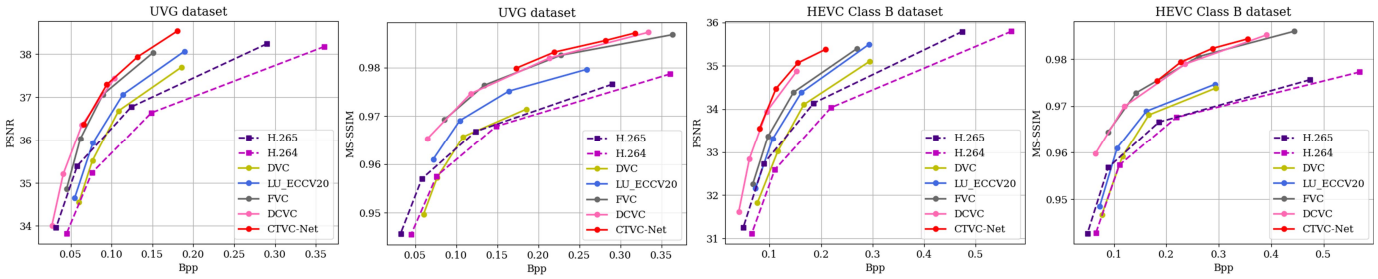
TABLE I
BDBR(%) Comparisons, Where H.265 is Employed as an Anchor.

	BDBR (PSNR) ↓			BDBR (MS-SSIM) ↓		
	UVG	HEVC B	MCL-JCV	UVG	HEVC B	MCL-JCV
H.264 [1]	35.27	28.12	31.35	20.06	16.81	18.99
DVC [4]	8.45	4.85	13.94	17.29	5.35	22.70
H.265 [2]	0	0	0	0	0	0
LU_ECCV20 [6]	-7.34	-15.92	4.75	-27.57	-10.58	5.02
FVC [5]	-28.71	-23.75	-21.08	-49.14	-53.97	-52.45
DCVC [8]	-35.00	-37.96	-23.08	-48.31	-50.72	-49.36
CTVC-Net(FP)	-36.62	-41.05	-25.11	-53.07	-58.05	-56.75
CTVC-Net(FXP)	-35.91	-40.32	-24.15	-52.13	-57.79	-55.96
CTVC-Net(Sparse)	**-35.19**	**-39.85**	**-23.44**	**-51.30**	**-57.11**	**-55.09**

Negative values of BDBR mean bit rate savings, while positive values mean more bit rate consumption.

C. Hardware Evaluation

To the best of our knowledge, there are **few dedicated ASIC-based accelerators for NVC**. Although some existing research effort to deploy inter-frame NVC on Snapdragon 8 chips [11] or FPGAs [12], direct comparisons are difficult due to platform variations and the absence of hardware performance metrics. Thus, to verify the hardware efficiency, we compare the NVCA with a widely-used commercial GPU product (NVIDIA RTX 3090), CPU product (Intel i9-9900X), and several ASIC-based pixel processing accelerators [25], [26]. In Table II, our design achieves 2.4× higher throughput and 799.7× better energy efficiency than the GPU, and 11.1× higher throughput and 1783.9× better energy efficiency than the CPU. Moreover, we surpass [25], [26] with up to **8.7× higher throughput and 2.2× better energy efficiency improvement**. The comparisons of average decoding time for 1080p videos between ours and other solutions [2], [5], [7]–[9] are listed in Fig. 9 (a). Notably, NVCA achieves a frame rate of 25 *FPS*, outperforming DCVC [8] by up to 22.7× in decoding speed. Besides, to validate the effectiveness of the heterogeneous layer chaining dataflow, we present a comparison of off-chip memory access (*GByte*) for different modules



(a) (b) (c) (d)

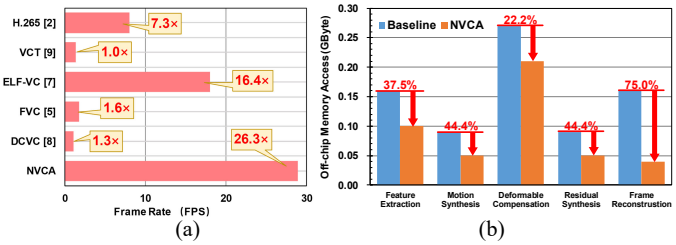
Fig. 8. RD-curves evaluated on PSNR and MS-SSIM metrics on the UVG and HEVC Class B datasets. (a) PSNR on the UVG dataset. (b) MS-SSIM on the UVG dataset. (c) PSNR on the HEVC Class B dataset. (d) MS-SSIM on the HEVC Class B dataset.

in Fig. 9 (b). We employ the layer-by-layer processing strategy as our baseline and achieve an overall **40.7% reduction** in the off-chip interaction compared to the baseline.

TABLE II
COMPARISON WITH OTHER PIXEL PROCESSING ACCELERATORS.

	CPU	GPU	[25]	Alchemist [26]	NVCA
Year	-	-	2022	2022	2023
Task	Video Compression		Feature Map Compression	Video Analysis	Video Compression
Benchmark	CTVC-Net	CTVC-Net	VGG16	VGG16	CTVC-Net
Technology (nm)	14	8	28	65	28
Frequency (MHz)	3500	1700	700	800	400
Precision (A-W)	FP 32-32	FP 32-32	FXP 16-16	FXP 16-16	FXP 12-16
Gate Count (M)	-	-	1.12	3.03†	5.01
On-Chip Memory (KB)	-	-	480	512	373
Power (W)	121.21	257.12	0.19	0.33†	0.76
Throughput (GOPS)	317	1493	403	833	**3525**
Energy Efficiency (GOPS/W)	2.6	5.8	2121.1	2524.2	**4638.2**

† They are the scale results from 65nm technology.



(a) (b)

Fig. 9. (a) Comparison of the decoding speed. (b) Comparison of the off-chip memory access.

VI. CONCLUSION

In this work, we have developed and validated an accelerator for NVC. At the algorithmic level, we propose a novel CTVC-Net and a fast algorithm-based sparse strategy to improve compression efficiency and reduce computational complexity. At the hardware level, we develop a sparse computing core and a heterogeneous layer chaining dataflow to flexibly support sparse Convs and DeConvs, significantly reducing off-chip interactions. Finally, the overall hardware architecture of NVCA is synthesized under the TSMC 28nm CMOS technology. Experimental results demonstrate that our design underscores superior improvements in video coding quality, decoding speed, and energy efficiency over prior arts.

REFERENCES

[1] T. Wiegand *et al.*, "Overview of the H.264/AVC video coding standard," *IEEE TCSVT*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[2] G. Sullivan *et al.*, "Overview of the high efficiency video coding (HEVC) standard," *IEEE TCSVT*, vol. 22, no. 12, pp. 1649–1668, Sep. 2012.

[3] B. Bross *et al.*, "Overview of the versatile video coding (VVC) standard and its applications," *IEEE TCSVT*, vol. 31, no. 10, pp. 3736–3764, 2021.

[4] G. Lu *et al.*, "DVC: An end-to-end deep video compression framework," in *CVPR*, Jun. 2019, pp. 10 998–11 007.

[5] Z. Hu *et al.*, "FVC: An end-to-end framework towards deep video compression in feature space," *IEEE TPAMI*, vol. 45, no. 4, pp. 4569–4585, Apr. 2023.

[6] G. Lu *et al.*, "Content adaptive and error propagation aware deep video compression," in *ECCV*, vol. 456-472, Aug. 2020, pp. 456–472.

[7] O. Rippel *et al.*, "ELF-VC: Efficient learned flexible-rate video coding," in *ICCV*, Oct. 2021, pp. 14 459–14 468.

[8] J. Li *et al.*, "Deep contextual video compression," in *NeurIPS*, Sep. 2021, pp. 18 114–18 125.

[9] F. Mentzer *et al.*, "VCT: A video compression transformer," in *NeurIPS*, 2022.

[10] G. Li *et al.*, "Block convolution: Toward memory-efficient inference of large-scale CNNs on FPGA," *IEEE TCAD*, vol. 41, no. 5, pp. 1436–1447, May 2022.

[11] H. Le *et al.*, "Mobilecodec: neural inter-frame video compression on mobile devices," in *MMSys*, Jun. 2022, pp. 324–330.

[12] C. Jia *et al.*, "FPX-NVC: an FPGA-accelerated P-frame based neural video coding system," in *VCIP*, Dec. 2022.

[13] W. Mao *et al.*, "FTA-GAN: A computation-efficient accelerator for GANs with fast transformation algorithm," *IEEE TNNLS*, vol. 34, no. 6, pp. 2978–2992, Jun. 2023.

[14] S. Zhang *et al.*, "An efficient accelerator based on lightweight deformable 3D-CNN for video super-resolution," *IEEE TCAS-I*, vol. 70, no. 6, pp. 2384–2397, Jun. 2023.

[15] X. Wang *et al.*, "WinoNN: Optimizing FPGA-based convolutional neural network accelerators using sparse winograd algorithm," *IEEE TCAD*, vol. 39, no. 11, pp. 4290–4302, Nov. 2020.

[16] J.-W. Chang *et al.*, "Towards design methodology of efficient fast algorithms for accelerating generative adversarial networks on FPGAs," in *ASP-DAC*, 2020, pp. 283–288.

[17] R. Zou *et al.*, "The devil is in the details: Window-based attention for image compression," in *CVPR*, Jun. 2022, pp. 17 471–17 480.

[18] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," in *ICCV*, Oct. 2021, pp. 9992–10 002.

[19] A. Lavin *et al.*, "Fast algorithms for convolutional neural networks," in *CVPR*, Jun. 2016, pp. 4013–4021.

[20] T. Xue *et al.*, "Video enhancement with task-oriented flow," *IJCV*, vol. 127, no. 8, pp. 1106–1125, Feb. 2019.

[21] A. Mercat *et al.*, "UVG dataset: 50/120fps 4k sequences for video codec analysis and development," in *MMSys*, May. 2020, pp. 297–302.

[22] H. Wang *et al.*, "MCL-JCV: a jnd-based h.264/avc video quality assessment dataset," in *ICIP*, Sep. 2016, pp. 1509–1513.

[23] Z. Wang *et al.*, "Multi-scale structural similarity for image quality assessment," in *ACSSC*, vol. 2, Nov. 2003, pp. 1398–1402.

[24] Y. Zhao *et al.*, "Dnn-chip predictor: An analytical performance predictor for dnn accelerators with various dataflows and hardware architectures," in *ICASSP*, vol. 2, May. 2020, pp. 1593–1597.

[25] Z. Shao *et al.*, "Memory-efficient cnn accelerator based on interlayer feature map compression," *IEEE TCAS-I*, vol. 69, no. 2, pp. 668–681, Feb. 2022.

[26] Y. Wang *et al.*, "An efficient deep learning accelerator architecture for compressed video analysis," *IEEE TCAD*, vol. 41, no. 9, pp. 2808–2820, Sep. 2022.