Configurable CNN Accelerator in Speech Processing based on Vector Convolution

Lanqing Hui, Shan Cao *Member, IEEE,* Zhiyong Chen, Shan Li, and Shugong Xu, *Fellow, IEEE*

***Abstract*—In speech applications, both input feature maps (IFMs) and kernels of neural networks are greatly diverse in shapes and sizes, which poses significant challenges to hardware acceleration. In this paper, a configurable CNN accelerator is introduced to make a good balance between the flexibility and efficiency for various neural network models in speech processing. The vector convolution scheme is first proposed by re-arrangement of IFM rows and weight values in vectors, by which the element convolution is converted into vector operations to break the limit of kernel-centric processing. The structure of vector processing element (VPE) is introduced to fit the continuous scaling down of IFMs with little control overheads, and the architecture of the CNN accelerator is proposed accordingly. FPGA implementation results demonstrate that the throughput is increased by 86% by the proposed architecture compared to state-of-the-art FPGA accelerators for the VGG16 network, while high DSP utilization is guaranteed for both 1D and 2D CNNs with various input sizes.**

***Index Terms*—Accelerator, CNN, speech processing, FPGA implementation**

I. Introduction

Speech processing techniques, such as automatic speech recognition (ASR) [1], speaker verification [2] and keyword spotting [3], are critical techniques in various fields of human-computer interaction. Many advanced solutions based on deep learning have been introduced in speech applications in recent years. Compared with convolutional neural networks (CNN) deployed in image processing, there are two main differences for CNNs in speech processing. Firstly, the inputs in speech processing have variable lengths due to the ever changing speaking rates and lengths of sentences. Secondly, both 1D and 2D convolutions are commonly used in speech processing.

In current CNN accelerators, computing resources are generally organized in 2D processing arrays to support common matrix–vector multiplications in 2D convolutions [4]–[6]. However, it may greatly decrease the utilization of computing and storage resources when migrated to 1D convolutions or convolutions with variable-length inputs.

To address the above issues, a high-throughput configurable architecture for neural network processing is proposed to accelerate the feature extraction of speech applications. The key features and main contributions of the proposed architecture are as follows.

L. Hui, S. Cao, Z. Chen, S. Li and S. Xu are with School of Communication and Information Engineering, Shanghai University. *Corresponding author: Shan Cao (Email: cshan@shu.edu.cn.)*

This work is supported by National Natural Science Foundation of China (No.61904101) and Shanghai Committee of Science and Technology of China (No.21ZR1422200).

- The row-based vector convolution mechanism is proposed to adapt both 1D and 2D convolutions with high utilization.
- The structure of vector processing element (VPE) is introduced which enables multiple dataflow organizations and hierarchical parallelism configuration. Extremely high resource utilization therefore could be achieved for various sizes of feature maps and filter kernels.
- The CNN acceleration architecture based on VPE arrays is proposed that high throughput and DSP efficiency is demonstrated for mainstream network models in speech processing.

The rest of the paper is organized as follows. In Section II, the motivation of row-based processing and the vector convolution is introduced. The overall architecture of the accelerator is proposed in Section III. Section IV presents FPGA implementation results of the proposed accelerator, and finally Section V concludes the paper.

II. Vector Convolution Scheme

A. Design Issues for Convolutions in Speech Processing

In speech applications, the raw input is generally a variable-length time-domain speech sequence as shown in Fig.1(a). Both 1D and 2D CNNs are commonly deployed as the backbone network. In 1D CNNs, the input is a $T-$length speech signal where T is time variable due to the change of speaking rates and lengths of sentence. As illustrated in Fig.1(c), after a 1D convolution layer with kernel size $K \times 1$ and pooling size $P \times 1$, the feature map is scaled to the length of T'. After several convolutional layers for feature extraction, fully connected (FC) layers are followed to generate global feature representations in the label space.

For 2D CNNs, the input speech signal is first converted to its spectrogram by Fourier transform, the size of which is $T \times F$. Here T is the length of speech and F is the range of frequency. As illustrated in Fig.1(d), several 2D convolution layers and pooling layers are performed which scaled the feature map to $T' \times F'$, and FC layers follows finally for network outputs.

If we consider 1D convolution layers as the special case of 2D convolution layers that $F = 1$, we can conclude two main features of the convolutional layer in speech processing. Firstly, the length of input feature maps (IFMs) is time-variable, while both dimensions of IFMs scale down with the layers going deeper. Secondly, the size of kernels change greatly in different network models. Both features pose great challenges in the architectural design of neural network accelerators in speech processing.

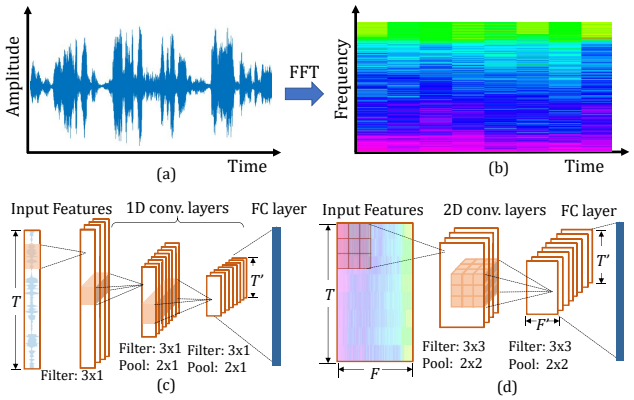

Fig. 1. Illustration of 1D and 2D CNN structures for speech processing. (a) The time domain waveform of a speech sequence; (b) the corresponding spectrogram of the speech sequence for figure (a) after FFT; (c) an example brief structure of a 1D CNN; (d) an example brief structure of a 2D CNN.

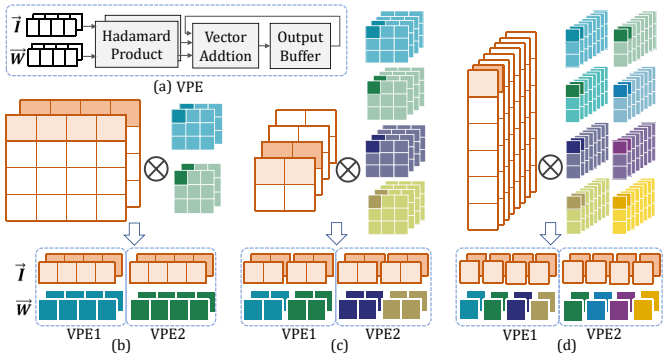

Fig. 2. Illustration of vector-based processing for 1D and 2D CNNs. (a) The block diagram of a vector-based processing element (VPE); (b) dataflow organization and data reuse for 2D convolutions with 4×4 feature maps and 3×3 kernels; (c) dataflow organization and data reuse for 2D convolutions with 2×2 feature maps and 3×3 kernels; (d) dataflow organization and data reuse for 1D convolutions with 6×1 feature maps and 3×1 kernels.

Currently, accelerators for CNNs are mostly kernel-centric, i.e., hardware resources are grouped according to the size of kernels. For instance, for row stationary dataflow architectures such as [5], the basic processing unit is essentially the element multiply-accumulation (MAC) unit for 2D convolutions. For [6], convolutions are accomplished by connected neighboring MAC units. On one hand, a convolution unit is generally designated for a specific kernel size, which may results in large amounts of idle resources with the changed kernel size. On the other hand, one dimension of IFMs may be scaled to 1 in speech processing, while the other dimension is time-variable. Therefore, the shape of IFMs vary dramatically, which may result in extreme low utilization of on-chip memory and computation resources.

B. Proposed Vector Convolution

We introduce the vector convolution mechanism to adapt both 1D and 2D convolutions with a wide range of IFM sizes and kernel sizes. For a 2D convolutional layer with $R \times S$ kernel size and C input channels, the output pixel is calculated by (1), where $I_{k,i,j}$ represents the pixel of the i-th row j-th column in IFM k, $W_{m,k,i,j}$ represents the filter weight value of the i-th row j-th column for input channel k output channel m, B_m is the bias of output channel m, and $O_{m,x,y}$ is the output pixel of the i-th row j-th column in OFM m after convolution.

$$O_{m,x,y} = \sum_{k=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{S-1} I_{k,x+i,y+j} \times W_{m,k,i,j} + B_m. \quad (1)$$

We consider the convolution of a vector, such as an IFM row symbolized by $\vec{I}_{k,x+i}$. We can calculate the convolution of a vector as shown in (2) by re-arrangement of operators and dataflow. Here, $\vec{a} \odot \vec{b}$ represents the Hadamard product of vectors $\vec{a}$ and $\vec{b}$, $\vec{a} \lhd j$ represents shifting vector $\vec{a}$ left by j elements. Weight vector $\vec{W}_{m,k,i,j} = \{W_{m,k,i,j}\}_Y$ is composed by Y copies of $W_{m,k,i,j}$, and bias vector $\vec{B}_m = \{B_m\}_Y$ by Y copies of B_m, where Y is the width of $\vec{I}_{k,x+i}$.

$$\vec{O}_{m,x} = \sum_{k=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{S-1} \vec{I}_{k,x+i} \odot \vec{W}_{m,k,i,j} \lhd j + \vec{B}_m. \quad (2)$$

In this way, the 2D convolution of an IFM is converted to Hadamard product and vector addition operations, instead of element MAC operations. As illustrated in Fig.2(a), a vector-based processing element (VPE) has n Hadamard product (HP) units, and a vector addition unit. n input channels can be processed in parallel to reduce the generation of partial sums.

For a 2D convolution as shown in Fig.2(b), there are 2 IFMs with size of 4×4 and 2×2 filters with size of 3×3. Assuming we have 2 VPEs working in single-instruction-multiple-data (SIMD), each VPE can be used for vector processing with maximum length of 4. A row of the IFM is fixed as $\vec{I}$, and a weight value in the filter is expanded to $\vec{W}$. A partial sum result of $\vec{O}$ is generated after the VPE processing. Then we turn to the next weight value until all weight values in a 3×3 filter are processed, and $\vec{I}$ is set to the next channel after that.

When the width of an IFM row scales as shown in Fig.2(c), we can set the output vector to $\vec{O}$ to $\{\vec{O}_{m,x}, \vec{O}_{m+1,x}\}$. In this way, input vectors would be re-organized and re-used, while the same hardware as shown in Fig.2(a) could be deployed in high utilization. As illustrated in Fig.2(c), $\vec{I}$ is composed of two copies of an input row, while $\vec{W}$ is the expanded by 2 weight values. Similarly, when the width of IFMs is scaled to 1, which is the case of 1D CNN, $\vec{O} = \{\vec{O}_{m,x}, \vec{O}_{m+1,x}, \vec{O}_{m+2,x}, \vec{O}_{m+3,x}\}$, i.e., four output channels of an IFM row are calculated simultaneously in a VPE.

III. System Architecture

Corresponding to the vector convolution scheme, a CNN accelerator architecture is introduced as illustrated in Fig.3. The CNN model is loaded as a series of instructions from the off-chip memory. Then the instruction is decoded in the Config Register into controlling information as illustrated in dotted lines. Input data is read through the DDR Controller and stored in the IFM buffer, weight buffers, and bias buffers, respectively. Then data is fetched to go through the VPE array for vector convolution and ReLu operations, through the Pooling unit for maxpooling (MaP) and global average pooling (GAP) operations, and through the FC unit for the FC layer.

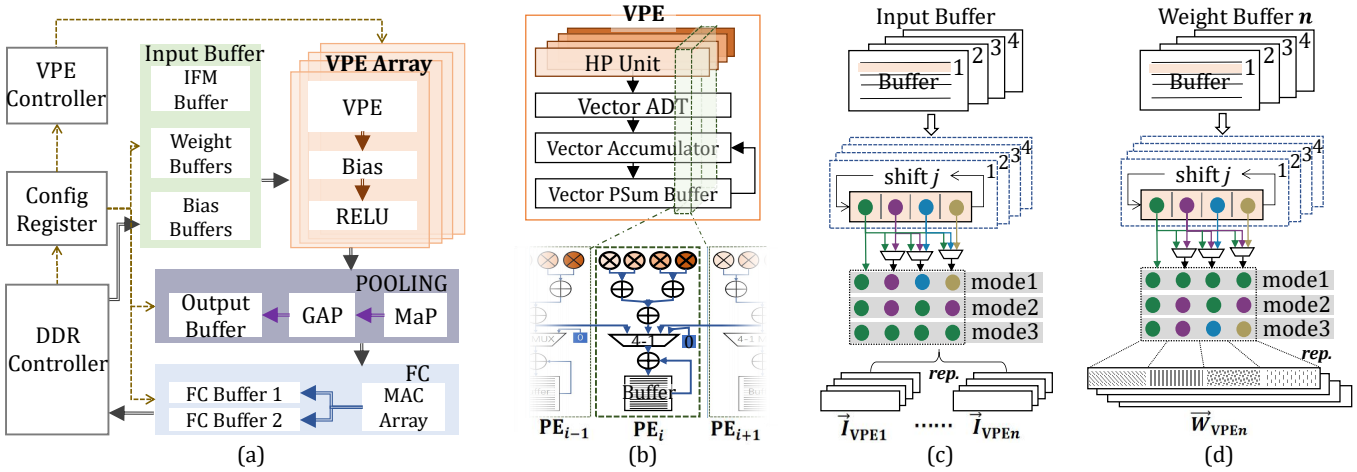

Fig. 3. The architecture of the proposed neural network accelerator for speech applications. (a) The overall framework of the accelerator; (b) the structure of a VPE; (c) the illustration of the IFM buffer which generates $\vec{I}$ for all VPEs; (d) the illustration of a weight buffer n which generates $\vec{W}$ for VPE n.

A. Parallelism in VPE Array

In the VPE array, there are N_{V} VPEs working in SIMD. The structure of each VPE is illustrated in Fig.3(b), integrating N_{HP} Hadamard product (HP) units to calculate the Hadamard product of two H-element vectors, a vector adder-tree (ADT) to sum up the results of the N_{HP} HP units, a vector accumulator and a vector partial-sum (Psum) buffer to accumulate and store the vector intermediate results.

The N_{HP} HP units are used for parallel processing of N_{HP} input channels, while N_{V} output channels are processed in parallel in the N_{V} VPEs. When the width of IFMs is W, $\lfloor H/W \rfloor$ output channels of an IFM row share an HP unit as shown in Fig.2. Therefore, the parallelism of output channels varies from N_{V} to $N_{\mathrm{V}} \times H$ according to the width of IFMs.

Since the structure of VPE is based on vector convolutions which are essentially accumulations of Hadamard products as stated in (2), all calculations in the VPE are therefore vector operations that multiple elements are processed in the same way to minimize controlling efforts. Only a multiplexer is required for each element to decide the input to the vector accumulator. a "0" padding is activated in case of data overlapping of different output channels.

B. Memory Hierarchy

Hierarchical memory structure is exploited in the proposed architecture, including local Psum buffer for intermediate data , input and output buffer for input data and output feature maps respectively, while other weight values and feature maps are stored in the off-chip memory.

In the IFM buffer as shown in Fig.3(c), there are N_{HP} row buffers corrsponding to the N_{HP} parallel input channels. $\lfloor H/W \rfloor$ rows from different output channels are combined and stored in a row buffer. The combined row is then read into a shift register. Each time, the leftmost row is read from the shift register, which is first expanded by $\lfloor H/W \rfloor$ times to form a complete vector, then copied by N_{V} times for all VPEs to perform Hadamard product with each filter. After that, the shift register is shifted by W elements so that the leftmost W elements make up a new physical row. In this way, reuses of IFMs varies according to their size , and the size of the IFM buffer and output buffer depends only on the vector size, not on the depth of IFMs, so that the changing length of speech would not bring memory resource overheads to the accelerator.

Weight buffers and bias buffers have similar structure as the IFM buffer as shown in Fig.3(d). In this way, the re-composition of input vectors is accomplished by shifting and copying, which is more scalable. Meanwhile, inputs to the following VPEs are in unified vector structure, which enables efficient data-driven processing.

IV. Implementation Results

The proposed accelerator is implemented on the Altera Arria 10 GX 1150 FPGA device, and compared with state-of-the-art neural network accelerators as shown in Table I.

A. Acceleration for typical backbone networks

Since few accelerators aim at CNNs in speech processing, we first implement the commonly-deployed backbone network VGG16 [12] to be compared with other accelerators. We set $H = 224$, $N_{\mathrm{HP}} = 2$, $N_{\mathrm{V}} = 8$. For Hadamard products, 3584 multipliers are required, among which 3036 multipliers are implemented by 1518 DSP blocks, while the remaining 548 multipliers are implemented by logic elements. Activations are quantized to 16 bits, while weights to 8 bits.

The corresponding implementation results and performance are listed in the 6-th column of Table I. The throughput is 1334.1 GOPS, and the latency per frame is 23.3 ms. Compared to the accelerator in [11] which deployed the same FPGA device as our work, the throughput is improved by 86%, and latency is reduced by 46%. We use DSP efficiency to represent the throughput achieved by each DSP block, which is 0.88 GOPS/DSP. However, since part of multipliers are implemented by logic elements, we consider the number of DSP blocks utilized as half of the multipliers required when calculating DSP efficiency for fair comparion, which is 0.74 GOPS/DSP as shown in Table I and still is the highest among related works.

TABLE I
FPGA IMPLEMENTATION RESULTS OF THE PROPOSED ACCELERATOR AND STATE-OF-THE-ART NEURAL NETWORK ACCELERATORS.

	[7]	[8]	[9]	[10]	[11]	ours		
FPGA	Xilinx XC7Z020	Xilinx Z-7045	Xilinx VCU440	Xilinx Z-7045	Arria 10 GX 1150	**Arria 10 GX 1150**	**Arria 10 GX 1150**	
DSP Utilization	190(86.4%)	855(95%)	1376(48%)	172(19%)	1518(100%)	**1518(100%)**	**128(8.4%)**	
Logic Utilization	29.9K(56%)	-	170K(6.7%)	-	138K(32%)	**163K(38%)**	**30K(7%)**	
Block Memory	85.5(61%)	-	1232(24%)	-	2232(82%)	**1872(69%)**	**310(11%)**	
Precision	8 bits	16 bits	16 bits	16 bits	16 bits	**8 bits(W)/16 bit(A)**	**8 bits(W)/16 bits(A)**	
Network	VGG16	VGG16	VGG16	VGG16	VGG16	**VGG16**	**RepVGG**	**X-vectors**
Input Size	224*224*3	224*224*3	224*224*3	224*224*3	224*224*3	**224*224*3**	**300*64*1**	**400*40**
Frequency(MHz)	214	125	200	140	200	**276**	**360**	**360**
GOP	30.95	30.95	30.76	30.95	30.95	**30.95**	**2.74**	**3.03**
Number of Weights	138.3M	138.3M	138.3M	138.3M	138.3M	**138.3M**	**8.7M**	**4.5M**
Latency(ms)	364	249.5	-	-	43.2	**23.3**	**15.2**	**23.3**
Throughput(GOPS)	84.3(Conv)	161.95	821	28.31	715.9	**1334.1**	**179.8**	**131.2**
DSP Efficiency(GOPS/DSP)	0.44(Conv)	0.19	0.60	0.17	0.47	**0.74**	**1.40**	**1.02**

B. Acceleration for networks in speech applications

In speech processing, two typical neural network models, X-vectors [2] and RepVGG [13], are deployed as example of 1D and 2D CNN models respectively. The setting of $H = 64$, $N_{\mathrm{HP}} = 2$ and $N_{\mathrm{V}} = 2$ is implemented accordingly, as listed on the rightmost two columns of Table I.

The clock frequency is increased to 360 MHz as the parallelism is decreased and the effect of global wiring is eliminated. For RepVGG,the DSP efficiency is 1.40 GOPS/DSP. For the first layer, the active DSP utilization rate is 50% since the number of input channel is 1, while for the remaining layers, the active DSP utilization rate is 100%. That is, as the size of feature map scales, VPEs still work in extreme high utilization. When this implementation is used for the 1D X-vector, the DSP efficiency is 1.02 GOPS/DSP, which is slightly lower. This is because there are two dilated convolutional layers in X-vectors. The utilization rate of DSP blocks are 40% and 50% for these two dilated layers, respectively, while instanced DSP blocks are 100% utilized for the remaining layers. Therefore, we can see that high utilization of computing resources is guaranteed as various input sizes for both 1D and 2D CNN models.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed the vector convolution scheme to guarantee high utilization for various input sizes. Two major changes are made. Firstly for the IFM data organization, the IFM-minimum-oriented dataflow combined with row-based convolution is deployed, so that the scale in row size can be compatible by reconfiguration while the scale in column size can be ignored. Then for the weight data organization, weights in a kernel are completely decoupled during convolution, so that the size of kernel could be largely configurable. To cover more operators in CNNs such as depthwise convolution, more complete instruction set and corresponding compiling techniques are necessary, which is part of our future work.

REFERENCES

[1] R. Haeb-Umbach, J. Heymann, L. Drude, S. Watanabe, M. Delcroix, and T. Nakatani, "Far-field automatic speech recognition," *Proceedings of the IEEE*, vol. 109, no. 2, pp. 124–148, 2021.

[2] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5329–5333.

[3] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted Residual Learning for Efficient Keyword Spotting," 2021. [Online]. Available: http://arxiv.org/abs/2106.04140

[4] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A Survey of Accelerator Architectures for Deep Neural Networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[5] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[6] Y. Parmar and K. Sridharan, "A Resource-Efficient Multiplierless Systolic Array Architecture for Convolutions in Deep Networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 2, pp. 370–374, 2020.

[7] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[8] S. I. Venieris and C. Bouganis, "FPGAConvNet: Mapping regular and irregular convolutional neural networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 326–342, Feb 2019.

[9] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, "Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, p. 97–106.

[10] H. Jia, X. Zou *et al.*, "An FPGA-Based Resource-Saving Hardware Accelerator for Deep Neural Network," *International Journal of Intelligence Science*, vol. 11, no. 02, p. 57, 2021.

[11] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

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

[13] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "RepVGG: Making VGG-style ConvNets Great Again," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 733–13 742.