VOTA: A Heterogeneous Multicore Visual Object Tracking Accelerator Using Correlation Filters

Junkang Zhu, *Graduate Student Member, IEEE*, Wei Tang, *Member, IEEE*, Ching-En Lee, Haolei Ye, Eric McCreath, *Member, IEEE*, and Zhengya Zhang, *Senior Member, IEEE*

***Abstract*— A correlation filter (CF) is a lighter and faster solution for visual object tracking (VOT) compared with a deep neural network (DNN). However, the CF-based trackers introduce diverse computational kernels and require FP16 computations. To address these challenges, we present a VOT accelerator (VOTA), a domain-specific accelerator for CF-based VOT that meets real-time performance at high efficiency, while providing flexibility and programmability. The VOTA encompasses a Winograd convolution core (WINO), a fast Fourier transformation (FFT) core (FFT), and a vector core (VEC) for diverse kernels, integrated in a high-bandwidth star-ring topology. The VOTA's frame-based instruction set and execution enable a 537 GFLOPS performance, adapt to variances of CF trackers, and reduce the code size. An instruction-chaining mechanism permits inter-core pipelining and improves the hardware utilization up to 84.2%. A 10.2-mm^2 28-nm FP16 system on chip (SoC) prototype incorporating the VOTA, an RISC-V host CPU, and other supportive peripherals is taped out and measured to achieve 2.45 TFLOPS/W at 0.72 V. Running the oriented particle CF (OPCF), a CF tracker enhanced by adaptive boosting and particle filtering, the SoC chip achieves 1157 frames/s (640 × 480 frame size) at 0.9 V and 175 MHz, consuming 296 mW.**

***Index Terms*— Correlation filter (CF), multicore accelerator, pipelining, visual object tracking (VOT), Winograd convolution.**

I. Introduction

VISUAL object tracking (VOT) is a computer vision task. Given the first frame in a video and the initial position of a target object, VOT obtains the positions of the object in the subsequent video frames. VOT finds practical applications in surveillance [1], transportation [2], robotics [3], and human–computer interface [4].

Manuscript received 17 January 2022; revised 25 March 2022 and 14 April 2022; accepted 14 April 2022. Date of publication 6 May 2022; date of current version 24 October 2022. This article was approved by Associate Editor Vivek De. This work was supported in part by the Ford-University of Michigan Research Alliance Research Alliance and in part by the Taiwan Semiconductor Manufacturing Company University Shuttle Program. *(Corresponding author: Junkang Zhu.)*

Junkang Zhu, Wei Tang, and Zhengya Zhang are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: jkzhu@umich.edu; weitang@umich.edu; zhengya@umich.edu).

Ching-En Lee was with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA. He is now with Temasek, Shanghai 200040, China.

Haolei Ye is with the Research School of Computer Science, Australian National University, Canberra, ACT 0200, Australia (e-mail: haolei.ye@anu.edu.au).

Eric McCreath was with the Research School of Computer Science, Australian National University, Canberra, ACT 0200, Australia. He is now with Skykraft Pty Ltd., Braddon, ACT 2612, Australia.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2022.3169946.

Digital Object Identifier 10.1109/JSSC.2022.3169946

Many recent VOT approaches have been designed based on deep neural networks (DNNs). Examples of DNN-based trackers include MDNet [5], TCNN [6], and DNT [7], which all apply a tracking-by-detection approach. However, a DNN-based tracking by detection approach is expensive, requiring on the order of 10 GOPS of computing for a low-resolution input frame and accessing on the order of 100 MBs of weight storage. For example, DNT [7] uses VGG16 [8] as its backbone, which requires 36 GOPS of computing over each 224 × 224 RGB image and 276 MBs of weight access. These translate to a high energy cost per frame. Online adaptation by training of DNN is also impractical, given the costly DNN training computation and long latency.

Correlation is a classic approach for signal detection and pattern matching, and it has been widely used in signal processing [9], [10], scientific experiments [11], and biomedical imaging [12]. Bolme *et al.* [13] pioneered the correlation approach in VOT by inventing the correlation filter (CF). In the 2014 VOT Challenge [14], three CF trackers took the top spots in performance. Since then, CF trackers continue to be among the top performers in the annual VOT Challenges [15]–[20].

CF tracks by correlating filters over an input, and the target is located by the maximum response [13], [21]. CF is commonly realized in the Fourier domain to simplify correlations with element-wise multiplications, and the filter adaption is done in parallel. Compared with DNN-based trackers, a CF tracker is a lighter and faster approach without performing detection on each frame. However, to support online training and Fourier-domain processing, a floating-point format, such as FP16, is required.

The state-of-the-art trackers combine convolutional neural network (CNN) and CF [22]–[25]. The rationale is that instead of operating on raw pixels, CF can operate in the feature space to improve its performance. For an edge platform, it is advantageous to combine a lightweight CNN front end with a CF back end to provide a good performance while keeping the complexity under control. Using this setup, a front-end CNN extracts features. For each feature, a CF is learned. To track a set of features associated with an object, multichannel CFs are adopted. After correlation, the multichannel responses are summed to locate the object.

In this work, we propose additional techniques to further improve the tracking performance and robustness, including adaptive boosting to adjust the weighting of each channel, and particle filtering [26], a Monte Carlo method, to narrow down

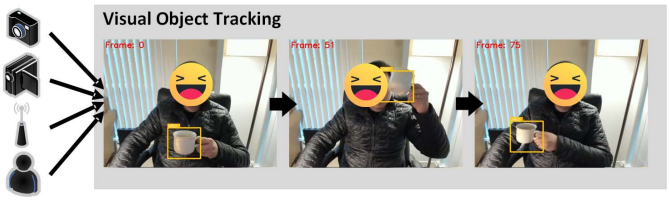


Fig. 1. Illustration of VOT: tracking a cup held by a hand.

the likely target regions to support fast motion. We name the algorithm oriented particle CF (OPCF).

Computationally, OPCF is a representative of an advanced CF tracker that requires diverse kernels, including convolution for CNN processing, fast Fourier transformation (FFT)/inverse FFT (IFFT) for conversion to Fourier domain for CF processing, and various real and complex vector operations. A custom application-specific integrated circuit (ASIC) is the most efficient way to implement OPCF with a fixed set of parameter choices. However, different problems require different sets of parameters and processing routines, and an ASIC lacks the flexibility to support a broad range of CF algorithms. A GPU provides the best flexibility, but it may not achieve the highest efficiency.

We present a programmable visual object tracking accelerator (VOTA) [27] to efficiently support the class of advanced CF trackers. VOTA employs three heterogeneous cores, a Winograd convolution core (WINO), an FFT core (FFT), and a vector core (VEC), all in FP16, to support the VOT computational kernels. The WINO is optimized for small-kernel convolutions; FFT is specialized for 2-D complex FFT/IFFT, and VEC provides flexible vector operations. VOTA adopts image-frame-based instructions to ease programming and reduce the code size. VOTA supports "chaining" of instructions via inter-core pipelining, allowing the hardware utilization to be maximized. VOTA is integrated with an RISC-V CPU, which acts as the host and provides the programmability.

The rest of this article is organized as follows. In Section II, we provide the background of VOT approaches. In Section III, we present OPCF, our benchmark workload. Section IV elaborates on the architecture of VOTA and its component designs. In Section V, we introduce the instruction-chaining techniques to improve the hardware utilization. Silicon measurement results and system evaluations are presented in Section VI. Section VII concludes this article.

II. CF Trackers for VOT

An example of VOT is shown in Fig. 1, where the initial location of the cup is given, and VOT tracks the location of the cup in each subsequent video frame highlighted in a bounding box.

In general, a CF is a trained feature filter. When the CF is applied to an input frame, it produces a response that corresponds to the location of the object. As an illustration, Fig. 2(a) shows an input target (t), a cup, and the CF (f) trained on tracking the cup. The input is correlated with the CF, and the output is a Gaussian response (r) that indicates the presence and the center location of the cup. Mathematically,

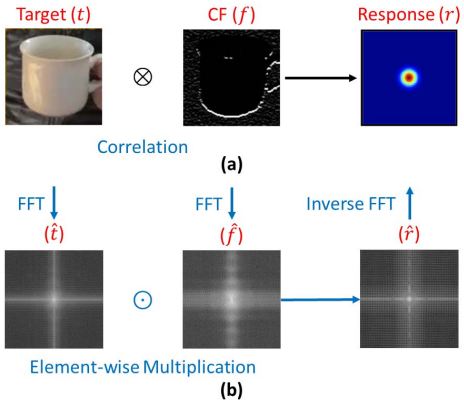


Fig. 2. Illustration of CF operation. (a) Correlation between an input target (t) and the CF (f) that produces a response (r). (b) Correlation performed in the Fourier domain.

the correlation is described by

$$r[i, j] = \sum_{p=-k}^{k} \sum_{q=-k}^{k} f[p, q]t[i + p, j + q]. \tag{1}$$

In CF, r, f, and t are all $(2k + 1) \times (2k + 1)$. The index of t wraps around when it overflows or underflows out of the range.

The correlation in the spatial domain can be computed more efficiently in the Fourier domain [13]. As illustrated in Fig. 2(b), correlations are computed by element-wise multiplications after the input and the CF are converted to the Fourier domain by 2-D FFT. The computation is described by

$$\hat{r} = \hat{f} \odot \hat{t}. \tag{2}$$

The spatial response can be recovered by an IFFT.

The Fourier domain computation also simplifies CF training, allowing CF training to be done alongside tracking, a significant advantage compared with DNN-based trackers. Early work by Bolme *et al.* [13] demonstrated online training of CF directly from the input video frames. Subsequently, histogram of oriented gradients (HOG) features [28] were used to train multiple CFs, forming the multichannel CF approach [29]. However, multichannel CF raises a robustness issue that less reliable channels may contribute to ambiguous responses. Some techniques were proposed to improve the reliability. CSRDCF [30] uses channel reliability scores to weigh channel responses. MCCT [23] maintains multiple experts with each one learning a certain combination of features, and it uses an adaptive switch to select the most reliable expert that yields the best performance for each frame.

The CF algorithm itself also evolved over the years. KCF [21] applies nonlinear kernels to CF training and inference to allow the tracker to adapt to a wider range of target variations. In more recent work, including DeepSRDCF [15], [31], C-COT [22], MCCT [23], CFCF [24], and MFT [25], CF trackers were combined with CNNs as dedicated feature extractors, because extracted features capture the target object more accurately than descriptor features [28] or hand-crafted features. CF has also been combined with particle filter [32] and Siamese network [33], [34] to improve the tracking robustness.

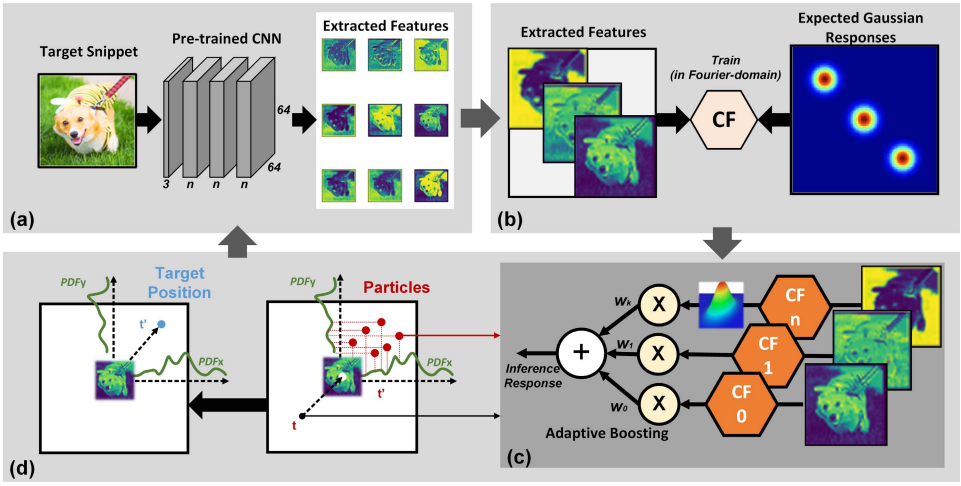


Fig. 3. Steps of the OPCF algorithm. (a) Feature extraction. (b) CF training. (c) CF inference. (d) Particle filtering.

Beginning in 2013 [35], the annual VOT Challenge ranks participating trackers by accuracy and robustness. CF trackers first made the top ranking in VOT 2014, and they have since dominated the top-10 ranking. CF trackers' excellent accuracy and robustness are largely due to their support of online training, a feature that DNN-based trackers lack due to the high cost of DNN training. Furthermore, even with online training, CF trackers generally require significantly less computation than entirely DNN-based trackers. In VOT 2016 [16], the top-two real-time bound trackers were CF trackers, Staple+ and SSKCF. Even with adaptive techniques to eliminate the need for complete online training, DNN-based trackers, such as [5]–[7], are among the slowest trackers [15], [16].

III. CF Tracker Design and Implementation Challenges

We create OPCF, a CF-based VOT algorithm for an edge platform. OPCF is designed based on the state-of-the-art CF trackers, and it captures their key computational features. OPCF serves as a representative of advanced CF trackers.

A. Algorithm Description

OPCF operates on a continuous video stream divided into 64×64 snippets. OPCF comprises four steps, feature extraction, CF training, CF inference, and particle filtering.

1) Feature Extraction: At the beginning of tracking, a snippet containing the target is selected from the first frame. Then, such as modern CF trackers [22], [23], [31], OPCF uses a pre-trained CNN as the front end to extract n features from the target. Unlike a deep CNN used for recognition, the CNN front end in a CF tracker is used for extracting features and denoise only; thus, it is much shallower than a typical deep CNN. A deep CNN is also not suitable for CF, because the deep layers cause features to lose spatial information of the target. A shallow CNN in a CF tracker also allows the computation to be kept lightweight. In this work, the CNN consists of three convolution layers, as illustrated in Fig. 3(a). An autoencoder [36] can also be used in place of the CNN to enable unsupervised training of the feature extractor.

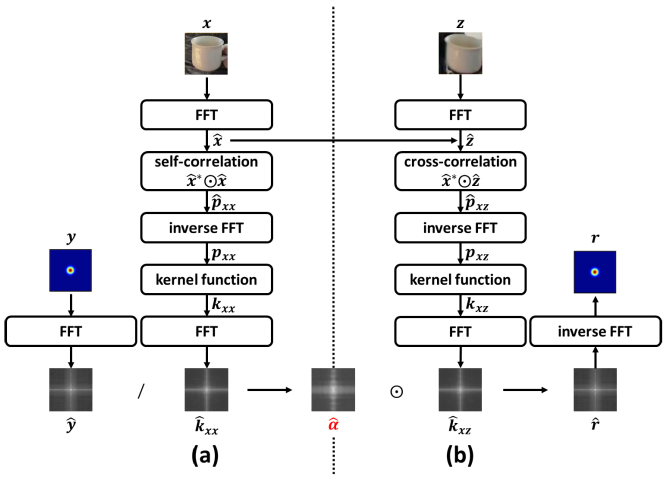


Fig. 4. Computational steps of (a) CF training and (b) CF inference.

2) CF Training: After feature extraction, a CF is trained for each of the n extracted features, as shown in Fig. 3(b), producing n CFs. The training computation is done in steps, as shown in Fig. 4(a), where x is the training input, y is the expected Gaussian response, and $\hat{\alpha}$ is the computed CF in the Fourier domain. CF training is performed in the Fourier domain, so a 2-D FFT is applied to x and y to obtain $\hat{x}$ and $\hat{y}$. The "kernel trick" from KCF [21] is adopted for better adaptation to nonlinear transformations of the target. In this work, we use a polynomial kernel to simplify the computation. In the Fourier domain, the self-correlation $\hat{p}_{xx}$ is computed first by element-wise multiplication, followed by computing the kernelized self-correlation $\hat{k}_{xx}$. Finally, the CF is computed by element-wise division of $\hat{y}$ by $\hat{k}_{xx}$. CF training is mathematically described by

$$\hat{\alpha} = \frac{\hat{y}}{\hat{k}_{xx} + \lambda} \tag{3}$$

where λ is a small value to avoid the divide-by-zero exception.

TABLE I
COMPUTATIONAL KERNELS OF AN OPCF TRACKER

Kernel	Feature Extraction	CF Training	CF Inference	Adaptive Boosting	Particle Filter
Convolution	✓				
FFT/Inverse FFT		✓	✓		✓
Real Vector Addition	✓	✓	✓	✓	✓
Real Vector Multiplication		✓	✓	✓	✓
Real Vector Division		✓			
Complex Vector Addition		✓	✓		✓
Complex Vector Multiplication		✓	✓		✓
Complex Vector Conjugate			✓		✓
Matrix Transpose		✓	✓		✓

3) CF Inference: The inference is first done on the image snippet at the location where the target is detected in the previous frame or manually specified (in the first video frame or upon a tracking failure). We use multichannel CF inference [29] where each CF trained is applied to an independent channel, producing n channels of output, as illustrated in Fig. 3(c). The steps of inference are shown in Fig. 4(b). z is the input, and r is the output. CF inference is also done in the Fourier domain. A 2-D FFT is first applied to z to obtain $\hat{z}$. The cross correlation with the training input $\hat{x}$ is computed by element-wise multiplication, followed by computing the kernelized cross correlation $\hat{k}_{xz}$. Finally, the inference is computed by element-wise multiplication with the CF $\hat{\alpha}$ and converted to the spatial domain. CF inference is mathematically described by

$$\hat{r} = \hat{k}_{xz} \odot \hat{\alpha}. \tag{4}$$

In practice, not all channels are of equal importance. According to the response quality of each CF, we assign a weight to reward strong CFs over time. The responses from channels are weighted and summed to produce the final response. The weights $\{w_i\}$ are learned by adaptive boosting [23], [30] and updated every time the target is successfully located. The approach boosts the contribution of reliable channels and suppresses unreliable ones. The final response peak is further checked against a threshold: if it is above a threshold, the target is located; otherwise, the target is missing, possibly due to fast motion.

4) Particle Filtering: To adapt to fast motion, we utilize particle filtering [26], a Monte Carlo method, to quickly narrow down the likely target regions using binary search. When the target is missing in the current snippet, additional snippets are sampled from the video frame according to the historical distribution of the target appearance. CF inference is applied to these additional snippets until the target is successfully located. Compared with a brute-force search, particle filtering uses binary search in the x- and y-directions to reduce the processing complexity from $O(n^2)$ to $O(\log n)$. Modern CF trackers such as MCPF [32] also adopted particle filtering.

After the target is located in the current frame, the CFs are trained and updated for inference for the next frame. The pseudocode of the OPCF algorithm is listed in Fig. 5.

1 Memory Space:
2 Correlation filters $\boldsymbol{\alpha} : \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$
3 Cached features $\boldsymbol{x} : x_0, x_1, \ldots, x_{n-1}$
4 Adaptive boosting weights $\boldsymbol{w} : w_0, w_1, \ldots, w_{n-1}$
5 For current frame F_t:
6 Select snippet from target position $F_{t,0}$;
7 Extract features $\boldsymbol{x} : x_0, x_1, \ldots, x_{n-1} = E(F_{t,0})$;
8 CF training $\boldsymbol{\alpha} : \alpha_0, \alpha_1, \ldots, \alpha_{n-1} = T(\boldsymbol{x}, y)$;
9 Update adaptive boosting weights $\boldsymbol{w}$;
10 Cache features $\boldsymbol{x}$;
11 For next frame $F_{t'}$:
12 Select snippet from last target position $F_{t',0}$;
13 Extract features $\boldsymbol{z} : z_0, z_1, \ldots, z_{n-1} = E(F_{t',0})$;
14 CF inference $\boldsymbol{r} : r_1, r_2, \ldots, r_{n-1} = I(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\alpha})$;
15 While weighted_sum$(\boldsymbol{w}, \boldsymbol{r}) >$ threshold:
16 Sample next probable snippet $F_{t',s}$;
17 Extract features $\boldsymbol{z} : z_0, z_1, \ldots, z_{n-1} = E(F_{t',s})$;
18 CF inference $\boldsymbol{r} : r_1, r_2, \ldots, r_{n-1} = I(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\alpha})$;
19 Update target position;
20 Repeat line 5 - 19;

Fig. 5. Pseudocode of the OPCF algorithm.

B. Implementation Challenges

Table I shows the level of diversity of computation kernels required by OPCF, which includes convolution, FFT/IFFT, vector operations, and matrix transpose. The diverse kernels present challenges for implementing CF trackers in hardware. General-purpose solutions, such as GPU, do not provide the best efficiency due to non-vectorized kernels such as FFT and complicated control overhead such as single instruction, multiple threads (SIMT) stack [37], operand collector [38], and caches [39], [40]. The existing ASICs only support a subset of kernels and lack flexibility [41]–[43]. In addition, CF trackers such as OPCF rely heavily on online training, which requires FP16 computations. FP16 is commonly not supported by inference ASICs.

To address these design challenges, a domain-specific accelerator for CF trackers is needed to meet real-time performance at high efficiency, while providing flexibility and programmability to accommodate variations of CF algorithms. The design requirements of such a domain-specific accelerator are as follows:

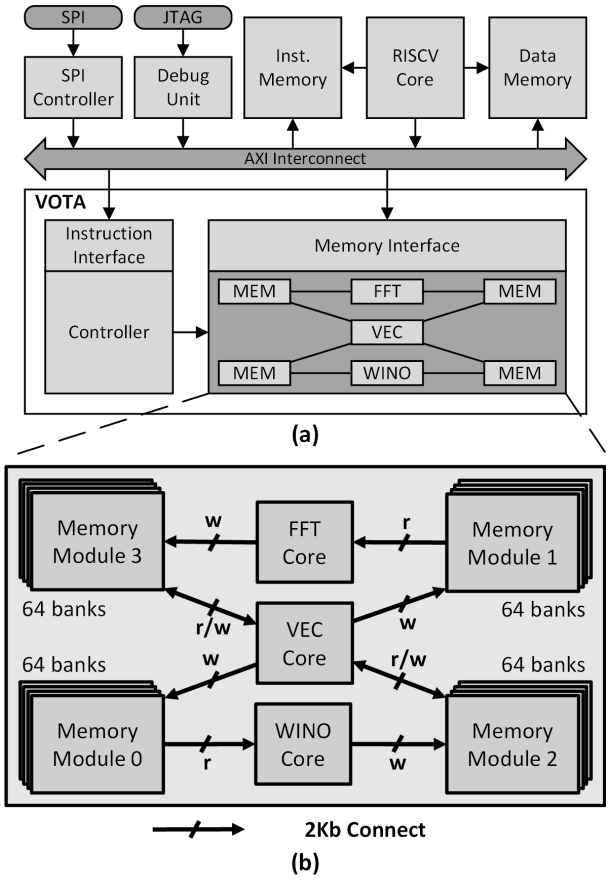


Fig. 6. (a) VOTA SoC system and (b) VOTA's multiple cores and memory modules in a high bandwidth star-ring topology.

1) provide multiple heterogeneous cores to support all computation kernels;
2) offer sufficient performance for real-time operations;
3) maintain high hardware utilization.

In the following, we describe the microarchitecture and hardware design features of VOTA to address these requirements.

IV. VOTA Microarchitecture Design

VOTA is a heterogeneous multicore accelerator made up of three functional cores and four shared 1-Mbit memory modules. The three functional cores are a WINO, an FFT, and a VEC, all in FP16, to support the diverse computational kernels for VOT. The WINO is optimized for small-kernel convolutions; the FFT is specialized for 2-D complex FFT/IFFT, and the VEC provides vector operations. The cores are instruction programmable. A controller decodes instructions to control signals to enable executions on the three cores.

VOTA is integrated into an open-source RISC-V platform, Pulpino [44], which includes an RISC-V CPU, an instruction memory, and a data memory over an Advanced Extensible Interface (AXI) interconnect, as shown in Fig. 6(a). The RISC-V CPU serves as the host to VOTA. The host sends instructions and inputs to VOTA through an instruction interface and a memory interface, respectively. The instruction interface is stream-based. It sets up the producer–consumer pair where the host produces and sends instructions continuously while VOTA receives and consumes them. The host provides the flow control of a program. The memory interface provides random access. The host can load data into any 32-bit memory word.

A. Star-Ring Topology

The cores and memory modules are connected using a high-bandwidth network-on-chip (NoC), as shown in Fig. 6(b). The NoC allows the three cores pass data through the four shared memory modules. Compared with a shared global bus that can be a communication bottleneck and a crossbar that cannot be implemented efficiently, we choose a star-ring topology to fit the common communication patterns between the cores in executing VOT programs.

Based on common communication patterns, VEC is placed in the center of the star to handle vector processing, shaping, passing, pre-processing, and post-processing for WINO and FFT. The read/write access to the memory modules is assigned, as shown in Fig. 6(b): the data flow is limited to one direction for WINO and FFT to reduce the communication bandwidth; VEC can initiate WINO and FFT processing by writing to the source memory modules and read/move the outputs of the two cores from the destination memory modules.

Each memory module supports wide word access with over 44-GB/s bandwidth to each core through multiple 2-Kbit-wide interconnects. A simple arbiter allows multiple cores to share a memory module in three access modes: 1) access a vector of 128 real numbers that corresponds to two rows in a 64 $\times$ 64 frame of real values; 2) access a vector of 64 complex numbers that corresponds to a row in a 64 $\times$ 64 frame of complex values; or 3) random access of any 32-bit word by the host.

B. Frame-Based ISA

Despite the diverse computational kernels, all the processing can be done on standard frames to streamline the data storage and flow. Accordingly, we define a frame-based instruction set architecture (ISA) for VOTA to simplify the programming of the three functional cores and reduce the code size.

A frame instruction operates on a frame of 64 $\times$ 64 real (FP16) or complex (2 $\times$ FP16) numbers. The format of a frame instruction is shown in Fig. 7(a). It consists of three parts: 1) opcode that specifies the operation; 2) destination that encodes the starting address where the output frame of the instruction is written to; and 3) source that encodes the starting address where the input frame is read from. Table II summarizes the frame instructions supported by WINO, FFT, and VEC. Two additional instructions, barrier (BARR) and halt (HALT), are included for flow control. The frame-based ISA significantly reduces the code size. Taking the example of OPCF, the frame-based ISA reduces the code size by 34$\times$ over a vector-based ISA, as shown in Fig. 7(b).

The three cores implement frame-based processing by first decomposing a frame instruction into row operations, and then executing the row operations by pipelined vector units. The process is demonstrated in Fig. 7(c), and it exploits two levels

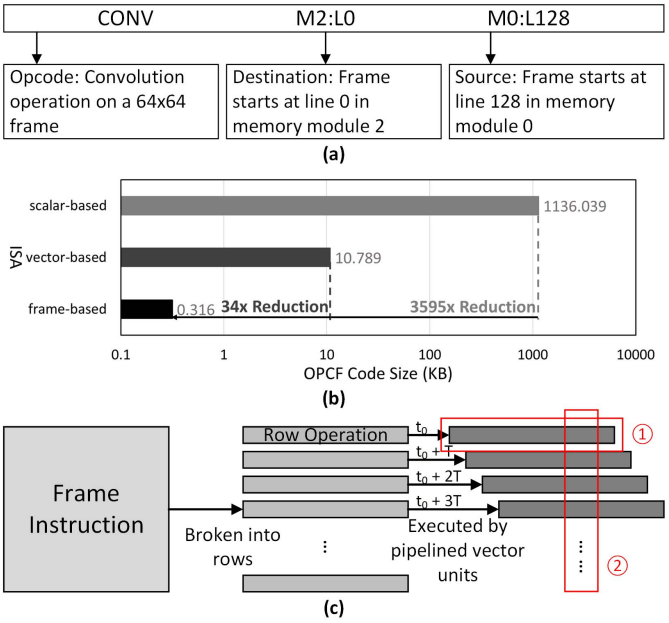


Fig. 7. VOTA's frame-based ISA. (a) Format of a frame instruction. (b) Reduction in code size by frame-based ISA over vector-based ISA. (c) Frame-based processing and two levels of parallelism.

TABLE II
VOTA'S FRAME-BASED ISA

Function Unit	Opcode	Description
WINO	CONV	convolution
FFT	FFT iFFT	FFT/inverse FFT
VEC	rADD rSUB rMULT rDIV rSQ	real number arithmetic
	rMV cMV	move
	cADD cSUB cMULT cREAL cIMAG cCONJ	complex number arithmetic
CONTROL	BARR	barrier
	HALT	halt

of parallelism: 1) parallel processing of numbers in a row ① and 2) parallel processing of multiple rows ②.

C. Functional Cores

The three functional cores, WINO, FFT and VEC, were designed for frame-based processing.

1) WINO: It adopts the Winograd algorithm [45] as an efficient way to compute 3 × 3 convolutions that are popular in the state-of-the-art CNNs. The comparison between a

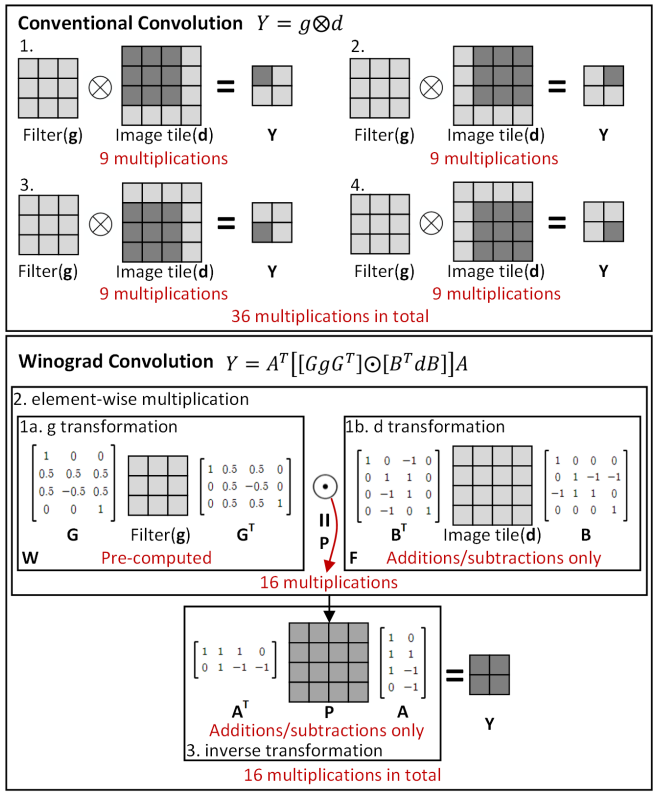


Fig. 8. Comparison between the conventional convolution and the Winograd convolution.

conventional convolution and a Winograd convolution is presented in Fig. 8. In this example, the conventional convolution computes a 3 × 3 filter g by a 4 × 4 input d convolution in a sliding window manner and generates a 2 × 2 output Y. In the Winograd convolution, the 3 × 3 kernel g is first transformed to a 4 × 4 weight matrix W in pre-computation. Then, three more steps follow: 1) d transform: the 4 × 4 input d is transformed to 4 × 4 matrix F; 2) element-wise multiplication: $P = W \odot F$; and 3) inverse transform: the 4 × 4 product P is transformed into a 2 × 2 matrix Y.

The Winograd convolution illustrated earlier uses 2.25× fewer multiplications than the conventional convolution. The reduction is attributed to two factors: 1) computing an element-wise multiplication $W \odot F$ in the Winograd convolution costs 16 multiplications, while computing a conventional convolution costs 36 multiplications, and 2) the transforms and inverse transforms in the Winograd convolution can either be pre-computed (i.e., g transform) or done using only additions and subtractions.

In VOTA, a WINO unit is designed to compute a 3 × 3 kernel by a 4 × 4 input tile convolution in a four-stage pipeline: d transform, element-wise multiplication, and two stages for the inverse transform. The WINO contains 32 WINO units to process 32 overlapping 4 × 4 input tiles at a time, constituting four input rows in a frame (with padding). The assignment of the input tiles is illustrated in Fig. 9(a). To support a stride-of-1 convolution, each tile overlaps with the neighboring tile by two rows/columns.

The inputs to the WINO units are through a set of input shift registers, as shown in Fig. 9(b). Reg 0 to Reg 3 buffer

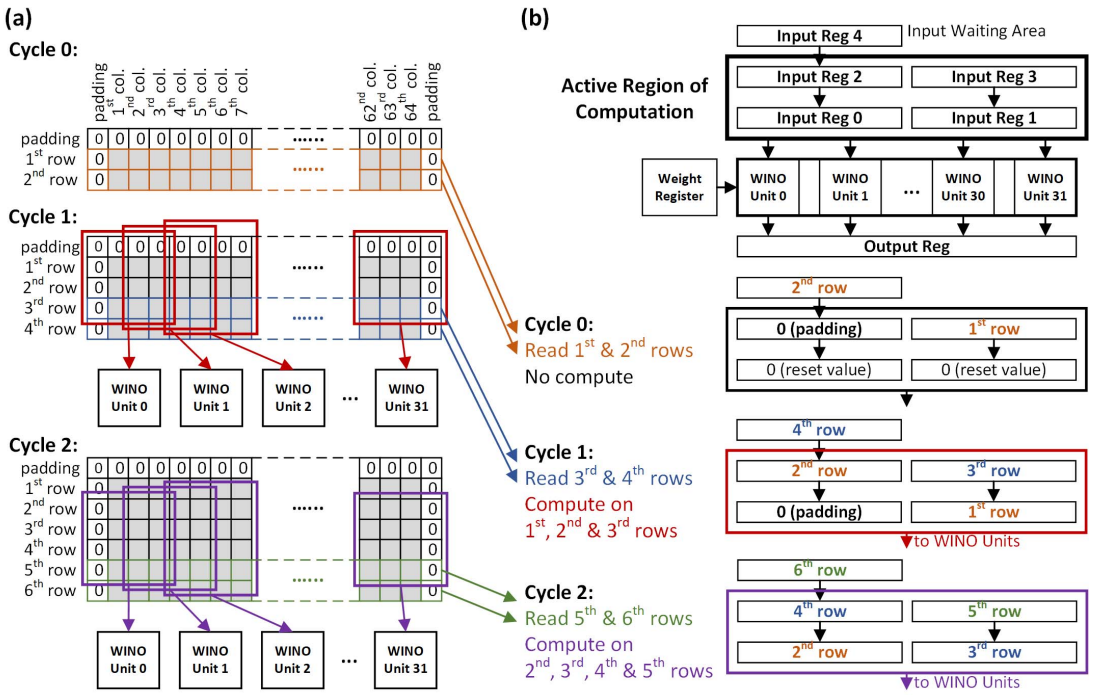


Fig. 9. Illustration of WINO operations. (a) Mapping input tiles to WINO units. (b) WINOs' operating steps.

the active region of computation. Reg 4 serves as the input waiting area. In every clock cycle, two rows of an input frame are fetched into Reg 3 and Reg 4, and the data buffered in the registers are shifted downward. After two cycles of initial loading, the active region contains four rows, and the 32 WINO units start to compute. WINO produces two output rows in each cycle. The output rows are buffered by an output register before being written back to a memory module. The input registers allow the input to be reused between adjacent WINO units and across clock cycles. A weight register stores the pre-computed weights W and broadcasts them to the WINO units to allow weight reuse.

WINO also supports kernels of other sizes: a 1×1 convolution can be realized by bypassing stages, and a 5×5 or larger convolution can be done by overlapping 3×3 convolutions.

2) FFT: It is designed to perform 2-D FFT and 2-D IFFT. Fig. 10(a) shows the composition of the FFT. The FFT vector machine is pipelined in seven stages for computing a 64-point complex FFT or IFFT. The matrix transpose required by 2-D operation is supported by a transpose buffer implemented in a bidirectional shift register array.

A 64×64 2-D FFT is computed in two passes through the FFT vector machine followed by a transpose after each pass, as shown in Fig. 10(b). Within each pass, 1-D FFTs are performed row-by-row by the FFT vector machine in a pipelined fashion. IFFT is computed by reversing the pipeline in the FFT vector machine to reuse the same set of compute units.

3) VEC: It is responsible for the all vector operations. It contains 64 VEC units, each made of a pipelined FP16 floating-point unit (FPU) that performs real and complex arithmetic operations. The supported operations are

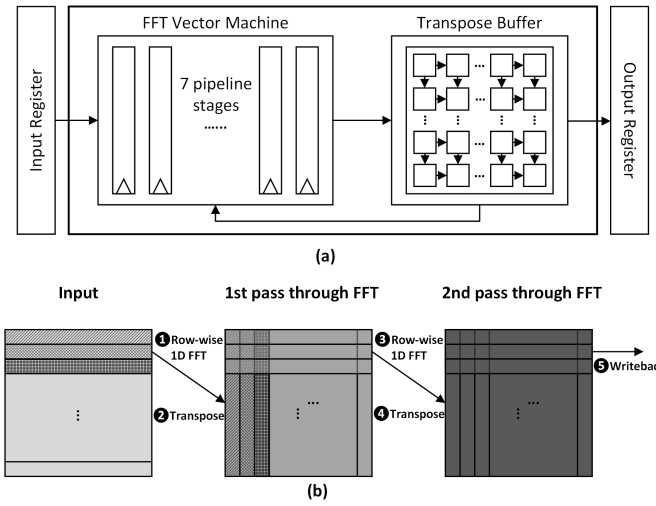


Fig. 10. (a) Design of FFT that contains a 64-point complex FFT vector machine and a transpose buffer. (b) Execution flow of a 2-D FFT.

summarized in the VEC section in Table II. The list includes real and complex addition (rADD/cADD), real and complex subtraction (rSUB/cSUB), real and complex multiplication (rMULT/cMULT), real division (rDIV), real number square (rSQ), and obtaining real and imaginary part (cREAL/cIMAG) and conjugate (cCONJ) for complex numbers. The VEC also handles direct data passing through move instructions for real and complex vectors (rMV/cMV).

V. Instruction Chaining and Inter-Core Pipelining

VOTA is a heterogeneous multicore accelerator, and the three functional cores can operate independently in parallel.

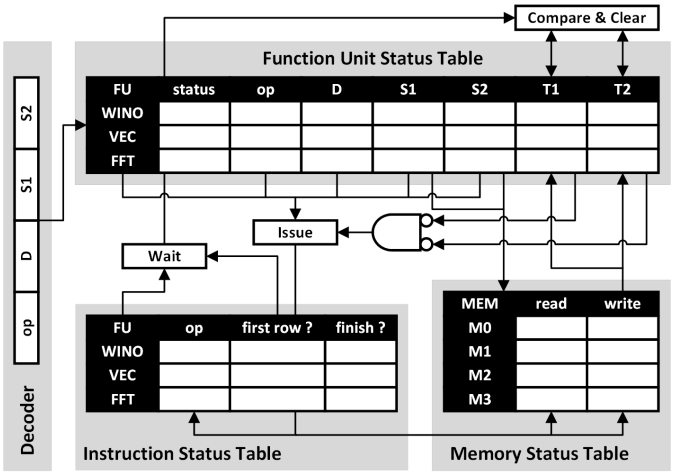


Fig. 11. Design of the instruction-chaining controller.

However, the VOT program's instructions are executed on all three cores, and the instructions that share data dependencies prevent the cores from running in parallel freely without data hazards. To solve data hazards, a naive solution is to insert a BARR instruction between any instruction pair with data dependency, but BARRs result in low hardware utilization, as a core needs to finish an entire frame instruction before the next core can proceed.

In VOTA, the frame-based processing and the row-by-row execution guide the design of each functional core and expose regular computational patterns and parallelism. This model also simplifies the tracking of data dependencies and instruction issuing. As each frame instruction is broken into row operations and each functional core produces one or multiple rows of results in each cycle, it is possible to pipeline the processing in units of rows, instead of frames, and allow frame instructions with dependencies to run on multiple cores in a fine-grained pipelined manner to improve core utilization.

To achieve this goal, VOTA adopts a hardware technique that uses an instruction-chaining controller to track run-time dependencies and issue instructions by exploiting inter-core pipelining. Design of the instruction-chaining controller is shown in Fig. 11. The instruction-chaining controller utilizes several scoreboards to track data dependency and resource availability to prevent data hazards. The functional unit status table (FUST) tracks the usage of the cores. Data dependencies are indicated by dependency tags T1 and T2, and the dependent data sources are recorded in S1 and S2. The memory status table (MST) tallies the availability of memory modules and controls the arbitration of the wide memory connects. The instruction status table (IST) informs the execution status of each active instruction using two flags to indicate the completion of the first row and the completion of the entire frame processing. An incoming instruction is first decoded and then registered in FUST. An instruction in FUST with both dependency tags T1 and T2 cleared can be registered in MST and IST and issued to a functional core to execute. The instruction-chaining controller has negligible cost and introduces only 0.005% and 0.029% overheads in area and power, respectively, based on synthesis results.

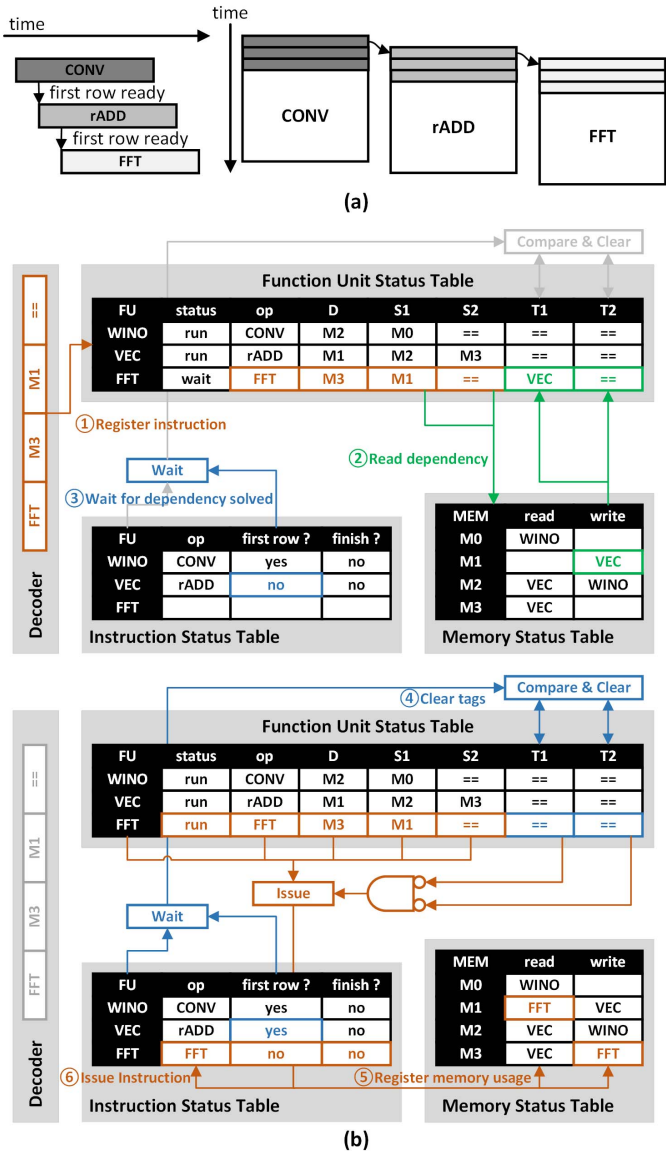


Fig. 12. (a) Execution flow of an instruction package with instruction chaining and inter-core pipelining. (b) Example of the execution of an instruction-chaining package.

To align with the instruction-chaining controller, we augment VOTA's frame-based ISA with a new structure called instruction-chaining package in which instructions with data dependencies are packed together and chained into an inter-core pipeline. An example is shown as follows:

```
PKG:
CONV    M2:L0 <= M0:L64
rADD    M1:L0 <= M2:L0 M3:L0
FFT     M3:L0 <= M1:L0.
```

In this instruction-chaining package, the second instruction (rADD) depends on the first instruction (CONV) as one input operand of rADD is the output operand of CONV. Similarly, the third instruction (FFT) depends on the second instruction (rADD). As illustrated in Fig. 12(a), with instruction chaining and inter-core pipelining, the downstream

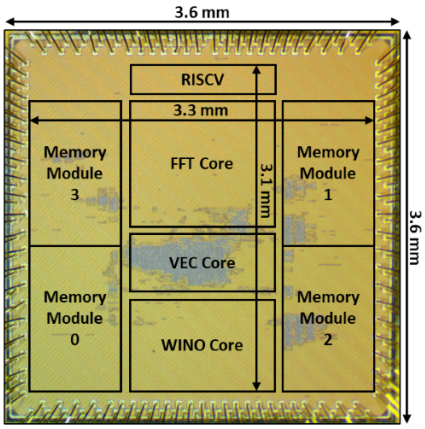


Fig. 13. Chip microphotograph.

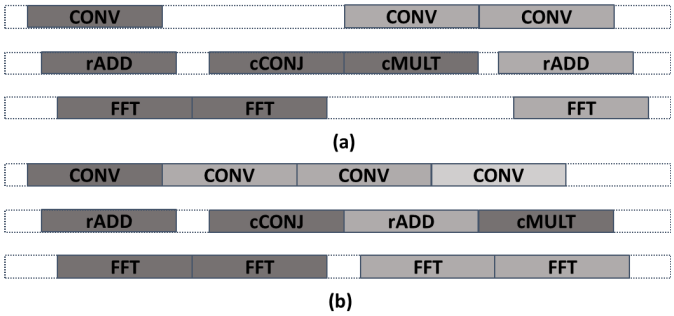


Fig. 14. (a) Execution flow of a standard instruction-chaining program (package by package). (b) Execution flow of an optimized program with package interleaving. The instruction chaining packages are coded in gray scales.

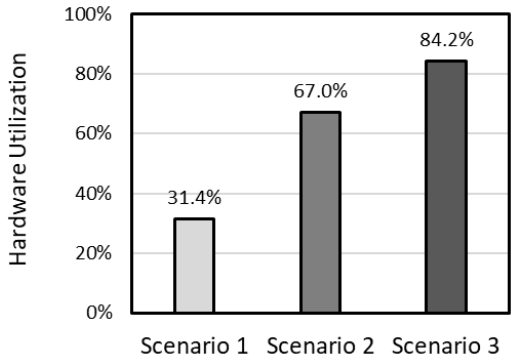


Fig. 15. Comparison of the hardware utilization of VOTA in three use scenarios. (Scenario 1: a standard program without instruction chaining; Scenario 2: a standard program with instruction chaining; Scenario 3: an optimized program with instruction chaining.)

instructions (rADD and FFT) do not need to wait for the entire frame to be completed by their upstream instructions (CONV and rADD, respectively) and can start whenever the first row is produced by the upstream instructions. The inter-core pipelining allows multiple cores to be active concurrently to achieve a high hardware utilization.

An example of executing the above instruction-chaining package is shown in Fig. 12(b). At the given time, the CONV and the rADD instructions are in the pipeline, the status of which are shown in IST: CONV has finished the first row, so the data dependency tag T1 for the VEC in FUST has been cleared, and rADD execution proceeds. Next, the FFT instruction enters ①. MST is read, and the data dependency between the FFT and the rADD instructions is found ②. As a result, the data dependency tag T1 for the FFT is set. The FFT waits for the VEC to finish the first row ③. After the VEC finishes the first row, the data dependency tag of the FFT is cleared ④, and the FFT instruction's source and destination memory modules are registered in MST ⑤. Now, the FFT instruction is ready to be issued ⑥. The procedure is followed for checking and clearing data dependency between instructions.

The instruction-chaining controller performs in-order instruction dispatch and issue, so the sequence of instructions presented to the controller can affect the level of parallelism the controller can see and exploit, which is characterized in Section VI-A.

An alternative approach to our hardware implementation of instruction chaining is with static scheduling by a compiler. The compiler needs to step through the cycle-by-cycle operating scenarios to establish the static scheduling, making the compilation more complicated. In comparison, our approach incurs a light hardware overhead, but the compiler can be made simple.

VI. Chip Measurement Results and Comparison

VOTA is integrated with an RISC-V core in a system on chip (SoC). The SoC was fabricated in a test chip in the TSMC 28-nm CMOS high performance compact mobile computing plus (HPC+) process. The test chip has a core area of 10.2 mm^2. A microphotograph of the chip is shown in Fig. 13. The chip contains a total of 3072 adders, 1536 multipliers, and 128 dividers, all in FP16, and 576 kB of SRAM as the instruction and data memory for the RISC-V core and the memory modules in VOTA.

A. Hardware Utilization Analysis

The effectiveness of instruction chaining and inter-core pipelining in improving the hardware utilization is evaluated in three use scenarios: 1) a standard OPCF program executed without instruction chaining; 2) a standard OPCF program executed with instruction chaining and inter-core pipelining enabled; and 3) an optimized OPCF program with optimal instruction sequence executed with instruction chaining and inter-core pipelining enabled.

The utilization in Scenario 1 is only 31%, because data dependency between instructions limits that only one core can execute most of the time. The same program is used in Scenario 2 where only adjacent instructions are packed in instruction-chaining packages. With instruction chaining and inter-core pipelining, the utilization is more than doubled to 67% as multiple cores can be active concurrently through fine-grained pipelining, as illustrated in Fig. 14(a). In Scenario 3, the hardware utilization can be further improved to 84% by optimizing the instruction sequence and interleaving instruction-chaining packages. It results in a more densely packed execution pipeline, as illustrated in Fig. 14(b).

TABLE III
MEASURED THROUGHPUT OF VOTA RUNNING RELEVANT BENCHMARKS

Benchmarks	OPCF	CF [13]	KCF [21]	VGG16 [8]
Throughput [FPS]	1157	1183	540	10

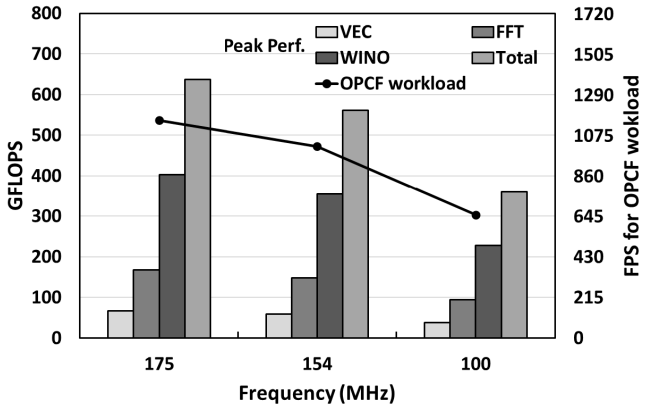


Fig. 16. Measured throughput of VOTA running OPCF at 0.9 V in room temperature.

Fig. 15 compares the hardware utilization of the three use scenarios.

B. Chip Performance Results

The performance of the chip is evaluated at 0.9 V and three different clock frequencies: 175, 154, and 100 MHz in room temperature. The performance of VOTA is measured by running OPCF as well as other VOT benchmarks, including a standard CF [13], a KCF [21], and VGG16 for DNN-based VOT [8]. The individual cores, WINO, FFT, and VEC, are evaluated by executing Winograd convolutions, 2-D FFTs, and vector-vector multiplications, respectively. In all evaluations, a FLOP is defined as an addition, a subtraction, a multiplication, or a division in FP16.

The measured throughput for executing OPCF is plotted in Fig. 16. The peak throughput of WINO, FFT, VEC, and VOTA (total) measured in FLOP-per-second (FLOPS) is shown in bars, and the throughput of executing OPCF measured in frames/s is shown in the solid line. At 0.9 V and 175 MHz, WINO, FFT, and VEC provide 403, 168, and 67 GFLOPS, respectively, which sum up to 638 FLOPS in total compute capacity. An optimized OPCF program operating on 640 × 480 inputs can utilize 537 FLOPS out of VOTA's compute capacity, enabling a 1157 frames/s throughput at a 0.86-ms latency. To demonstrate the versatility of VOTA, the throughput of the chip running relevant benchmarks is listed in Table III.

The measured power consumption of the chip with voltage and frequency scaling is shown in Fig. 17. At 0.72 V and 100 MHz, the chip achieves the peak power efficiency of 2.45 TFLOPS/W running the OPCF program, dissipating 124 mW. At the nominal voltage of 0.9 V, the chip consumes 296 mW at 175 MHz for a power efficiency of 1.81 TFLOPS/W.

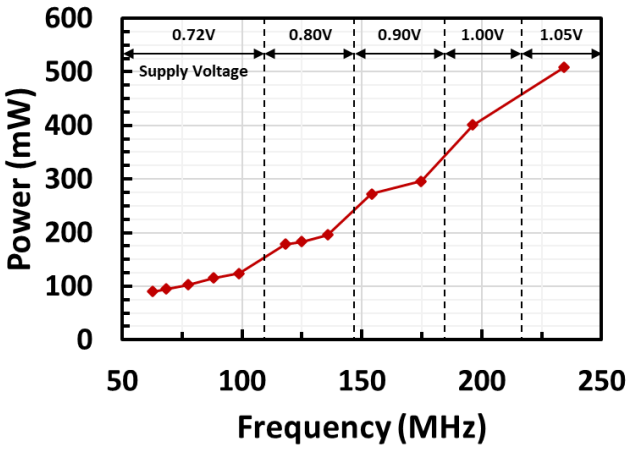


Fig. 17. Measured power consumption with voltage and frequency scaling.

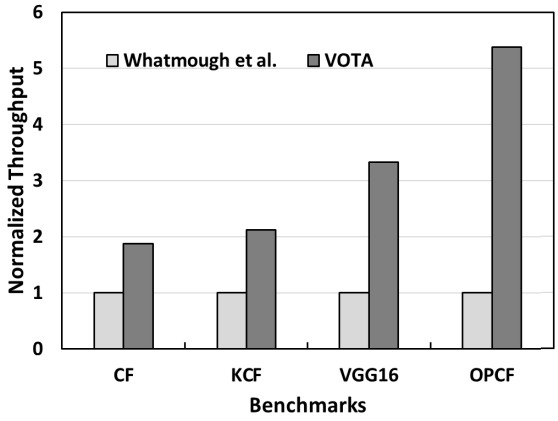


Fig. 18. Performance comparison with Whatmough *et al.* [46].

C. Comparison With Prior Work

VOTA is a programmable domain-specific accelerator for VOT to support advanced CF trackers. VOTA encompasses three heterogeneous cores connected in a high-bandwidth star-ring topology to support high-throughput computation involving different kernels. VOTA supports fine-grained inter-core pipelining to improve utilization. VOTA also enables adaptive online training with FP16 computations. Compared with recent DNN and vision processors, VOTA demonstrates a high power efficiency and a competitive compute density, as shown in Table IV.

Whatmough *et al.* presented a highly flexible heterogeneous SoC [46] for DNN and DSP that can potentially execute a CF tracker. In particular, the convolution kernel can be mapped to datapath accelerators and executed in a sliding window manner at high power efficiency. The FFT kernel can be mapped to eFPGA tiles. Within the eFPGA, DSP tiles can easily perform the FFT computation, and the interconnection networks in logic tiles can be customized for handling the FFT butterflies. Matrix transpose and other vector operations can

TABLE IV
COMPARISON WITH PRIOR WORK

Works	**VOTA This Work**	Whatmough [46]	Suleiman [47]	Oh [48]	Lee [49]
Application	**Visual Object Tracking**	DNN, DSP, Security	Visual-inertial Odometry	DNN training	DNN training
Architecture	**Programmable SoC**	Programmable SoC	ASIC	ASIC	ASIC
Technology [nm]	**28**	16	65	14	65
Data type	**FP16**	-	FXP[(b)]/FP[(c)]	FP16	FP16
Area [mm^2]	**10.23**	25	16.07	9.844	16
SRAM	**576 KB**	9 MB	854 KB	2 MB	372 KB
Core Voltage [V]	**0.72 - 1.05**	0.5 - 1	1	0.54 - 0.62	0.78 - 1.1
Frequency [MHz]	**60 - 230**	353.8 - 732.48	62.5[(b)]/83.3[(c)]	1000 - 1500	50 - 200
Power [mW]	**90 - 508**	-	27	-	43.1 - 367
Throughput [GOPS or GFLOPS]	**193 - 720**	10 - 100[(a)]	10.5 - 59.1[(b)]/1 - 5.7[(c)]	2000 - 3000	>300
Area efficiency [GOPS/mm^2 or GFLOPS/mm^2]	**70.38**	4	4.03	304.75	18.75
Power efficiency [TOPS/W or TFLOPS/W]	**2.45**	1.04	2.4[(d)]	1.4	1.74

[(a)] Derived from a figure. [(b)] Fixed-point in VFE. [(c)] Floating-point in BE. [(d)] Estimated peak power efficiency.

also be realized in logic tiles. Alternatively, vector operations can be executed by the ARM CPUs or by the single instruction, multiple data (SIMD) units. The throughput of the datapath accelerators and the throughput of the eFPGA tiles are both around 100 GOPS, lower than the WINO and the FFT in VOTA. Furthermore, the bit width of the on-chip data link is limited to 128 bit, which limits the total throughput of the chip to 40 GOPS if inter-core pipelining is adopted. For this reason, activating cores in series is the optimal way to run a CF tracker in [46]. Fig. 18 shows the performance of VOTA compared with the estimated performance in [46] for VOT benchmarks. VOTA outperforms [46], especially in OPCF where the kernels are diverse and heavy in computation. Finally, [46] only supports integer operations, which is not suitable for advanced CF trackers that require Fourier-domain processing and online training.

Navion [47] is a heterogeneous ASIC designed for visual-inertial odometry (VIO)-based simultaneous localization and mapping (SLAM) for autonomous navigation. It consists of two front-end modules (VFE and IFE) that process camera and sensor inputs and a back-end module (BE) that solves equations and produces the object's trajectory. Navion adopts clock gating and fixed-point arithmetic in VFE for high energy efficiency. However, Navion's lack of instruction programmability and inflexible processing pipeline, and lack of FP16 support make Navion unsuitable as a domain-specific accelerator.

VII. CONCLUSION

We present VOTA, a programmable domain-specific accelerator for advanced CF trackers. VOTA consists of three FP16 cores: WINO, FFT, and VEC to support advanced CF trackers' diverse computational kernels. The cores are linked in a star-ring topology to enable high-bandwidth connectivity between the cores and to fit the communication patterns between the cores in executing VOTA programs. VOTA is integrated in an RISC-V platform, and it uses the RISC-V CPU as the host in executing programs.

VOTA adopts frame-based ISA, and its instructions operate on standard 64×64 frames to simplify programming. Internally, VOTA cores decompose frames to rows to be executed by pipelined vector units. As a row is the smallest work unit, fine-grained inter-core pipelining between cores is made possible by chaining instructions to improve the hardware utilization.

A 10.2-mm^2 VOTA test chip was fabricated in the TSMC 28-nm CMOS HPC+ process. The chip is measured to provide 2.45 TOPS/W at 0.72 V and 100 MHz. When executing OPCF, it achieves a throughput of 1157 frames/s (640×480 frame size). Compared with a state-of-the-art SoC, VOTA's throughput is up to $5\times$ higher with more than twice the power efficiency.

REFERENCES

[1] A. Gautam and S. Singh, "Trends in video object tracking in surveillance: A survey," in *Proc. 3rd Int. Conf. I-SMAC (IoT Social, Mobile, Analytics Cloud) (I-SMAC)*, Dec. 2019, pp. 729–733.

[2] J. Yang, R. Xu, J. Cui, and Z. Ding, "Robust visual tracking using adaptive local appearance model for smart transportation," *Multimedia Tools Appl.*, vol. 75, no. 24, pp. 17487–17500, 2016.

[3] S. Li and D.-Y. Yeung, "Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models," in *Proc. AAAI Conf. Artif. Intell.*, vol. 31, Feb. 2017, pp. 1–7. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11205

[4] C.-L. Huang and C.-Y. Chung, "A real-time model-based human motion tracking and analysis for human-computer interface systems," *EURASIP J. Adv. Signal Process.*, vol. 2004, no. 11, pp. 1–15, 2004.

[5] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4293–4302.

[6] H. Nam, M. Baek, and B. Han, "Modeling and propagating CNNs in a tree structure for visual tracking," 2016, *arXiv:1608.07242*.

[7] Z. Chi, H. Li, H. Lu, and M.-H. Yang, "Dual deep network for visual tracking," *IEEE Trans. Image Process.*, vol. 26, no. 4, pp. 2005–2015, May 2017.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent., (ICLR)* San Diego, CA, USA, May 2015, pp. 1–14.

[9] H. Lai and W. Xu, "Robust kernel correlation based bi-channel signal detection with correlated non-Gaussian noise," *IEEE Signal Process. Lett.*, vol. 28, pp. 165–169, 2021.

[10] G. Zhang *et al.*, "Nonlinear processing for correlation detection in symmetric alpha-stable noise," *IEEE Signal Process. Lett.*, vol. 25, no. 1, pp. 120–124, Jan. 2018.

[11] P. Hu, L. Liu, and L. Shen, "The application of orthogonality cross correlation algorithm in weak signal detection," *J. Phys., Conf. Ser.*, vol. 1314, no. 1, Oct. 2019, Art. no. 012154, doi: 10.1088/1742-6596/1314/1/012154.

[12] P. Leydon, M. O'Connell, D. Greene, and K. M. Curran, "Cross-correlation template matching for liver localisation in computed tomography," in *Proc. Irish Mach. Vis. Image Process. Conf. (IMVIP)*, 2019, pp. 1–6.

[13] D. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 2544–2550.

[14] M. Kristan *et al.*, "The visual object tracking VOT2014 challenge results," in *Proc. Comput. Vis. ECCV 2014 Workshops*, L. Agapito, M. M. Bronstein, and C. Rother, Eds. Cham, Switzerland: Springer, 2015, pp. 191–217.

[15] M. Kristan *et al.*, "The visual object tracking VOT2015 challenge results," in *Proc. IEEE Int. Conf. Comput. Vis. Workshop (ICCVW)*, Dec. 2015, pp. 564–586.

[16] M. Kristan *et al.*, "The visual object tracking VOT2016 challenge results," in *Proc. Comput. Vis. ECCV Workshops*, G. Hua and H. Jégou, Eds. Cham, Switzerland: Springer, 2016, pp. 777–823.

[17] M. Kristan *et al.*, "The visual object tracking VOT2017 challenge results," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2017, pp. 1949–1972.

[18] M. Kristan *et al.*, "The sixth visual object tracking VOT2018 challenge results," in *Proc. Comput. Vis. ECCV Workshops*, L. Leal-Taixé and S. Roth, Eds. Cham, Switzerland: Springer, 2019, pp. 3–53.

[19] M. Kristan *et al.*, "The seventh visual object tracking VOT2019 challenge results," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 2206–2241.

[20] M. Kristan *et al.*, "The eighth visual object tracking VOT2020 challenge results," in *Proc. Comput. Vis. ECCV Workshops*, A. Bartoli and A. Fusiello, Eds. Cham, Switzerland: Springer, 2020, pp. 547–601.

[21] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015.

[22] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking," in *Proc. Eur. Conf. Comput. Vis.*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 472–488.

[23] N. Wang, W. Zhou, Q. Tian, R. Hong, M. Wang, and H. Li, "Multi-cue correlation filters for robust visual tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4844–4853.

[24] E. Gundogdu and A. A. Alatan, "Good features to correlate for visual tracking," *IEEE Trans. Image Process.*, vol. 27, no. 5, pp. 2526–2540, May 2018.

[25] S. Bai, Z. He, T.-B. Xu, Z. Zhu, Y. Dong, and H. Bai, "Multi-hierarchical independent correlation filters for visual tracking," 2018, *arXiv:1811.10302*.

[26] P. Del Moral, A. Doucet, and A. Jasra, "Sequential Monte Carlo samplers," *J. Roy. Stat. Soc. B, Stat. Methodol.*, vol. 68, no. 3, pp. 411–436, 2006, doi: 10.1111/j.1467-9868.2006.00%553.x.

[27] J. Zhu, W. Tang, C.-E. Lee, H. Ye, E. McCreath, and Z. Zhang, "VOTA: A 2.45TFLOPS/W heterogeneous multi-core visual object tracking accelerator based on correlation filters," in *Proc. Symp. VLSI Circuits*, Jun. 2021, pp. 1–2.

[28] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, no. 1, Jun. 2005, pp. 886–893.

[29] H. K. Galoogahi, T. Sim, and S. Lucey, "Multi-channel correlation filters," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 3072–3079.

[30] A. Lukezic, T. Vojir, L. C. Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter with channel and spatial reliability," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4847–4856.

[31] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg, "Learning spatially regularized correlation filters for visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 4310–4318.

[32] T. Zhang, C. Xu, and M.-H. Yang, "Multi-task correlation particle filter for robust object tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4819–4827.

[33] S. Chen, D. Qiu, and Q. Huo, "Siamese networks with discriminant correlation filters and channel attention," in *Proc. 14th Int. Conf. Comput. Intell. Secur. (CIS)*, Nov. 2018, pp. 110–114.

[34] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with Siamese region proposal network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8971–8980.

[35] M. Kristan *et al.*, "The visual object tracking VOT2013 challenge results," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, Sep. 2013, pp. 98–111.

[36] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. ICML Workshop Unsupervised Transf. Learn., (PMLR)*, vol. 27, Jul. 2012, pp. 37–49. [Online]. Available: https://proceedings.mlr.press/v27/baldi12a.html

[37] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic warp formation and scheduling for efficient GPU control flow," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Dec. 2007, pp. 407–420.

[38] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2011, pp. 308–317.

[39] H. Igehy, M. Eldridge, and K. Proudfoot, "Prefetching in a texture cache architecture," in *Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop Graph. Hardw. (HWWS)*, 1998, p. 133, doi: 10.1145/285305.285321.

[40] H. P. Hofstee, "Power efficient processor architecture and the cell processor," in *Proc. 11th Int. Symp. High-Perform. Comput. Archit.*, 2005, pp. 258–262.

[41] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[42] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[43] J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 1–14.

[44] Pulp-Platform. *Pulp-Platform/Pulpino: An Open-Source Microcontroller System Based on RISC-V*. Accessed: Apr. 30, 2022. [Online]. Available: https://github.com/pulp-platform/pulpino

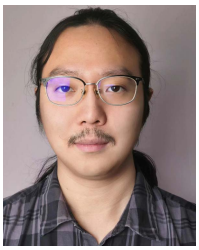
[45] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4013–4021.

[46] P. N. Whatmough *et al.*, "A 16nm 25 mm^2 SoC with a 54.5× flexibility-efficiency range from dual-core arm cortex-A53 to eFPGA and cache-coherent accelerators," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C34–C35.

[47] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 133–134.

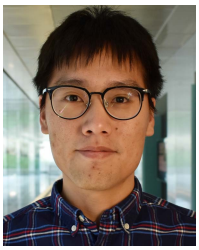
[48] J. Oh *et al.*, "A 3.0 TFLOPS 0.62 V scalable processor core for high compute utilization AI training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.

[49] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 LNPU: A 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 142–144.



Junkang Zhu (Graduate Student Member, IEEE) received the B.S. degree in physics from Nanjing University, Nanjing, China, in 2017, and the M.S. degree in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2019, where he is currently pursuing the Ph.D. degree in electrical and computer engineering.

His current research interests include novel microarchitecture, system-on-chip design, and system integration for machine learning and computer vision applications.



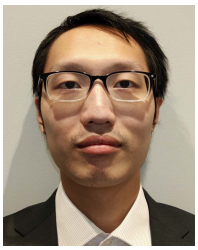
Wei Tang (Member, IEEE) received the B.S. degree from National Chiao-Tung University, Hsinchu, Taiwan, in 2011, and the M.S. and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, both in 2019.

He was a Visiting Ph.D. Researcher with Lund University, Lund, Sweden, and a Graduate Research Intern with Intel Laboratories, Hillsboro, OR, USA. He is currently an Assistant Research Scientist with the Department of Electrical Engineering and Computer Science, University of Michigan. His research interests include the high-speed, energy-efficient, and flexible VLSI designs for communications, machine learning, and robotics.

Ching-En Lee received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2012, the M.S. degree in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2015, and the Ph.D. degree in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2020.

From 2015 to 2016, he was with Intel Laboratories, Hillsboro, OR, USA, where he worked on real-time machine learning hardware acceleration for full-duplex radios. From 2018 to 2019, he was with Iluvatar Corex, San Jose, CA, USA, where he led the development of deep learning inference and training systems on chip (SoCs) and for edge to cloud computing applications. From 2020 to 2021, he was with McKinsey & Company, Shanghai, China, where he worked across the technology, media, and telecom (TMT), consumer and retail, and semiconductor sectors with functional emphasis on growth and transformation strategy, end-to-end business building, and product go-to-market. He is currently with Temasek, Shanghai, focusing on Tech Investment.

His past research interests include efficient domain-specific computing architectures and full-stack systems design for machine learning, deep learning, computer vision, and robotics.



Haolei Ye received the B.S. and M.S. degrees in software engineering from Australian National University, Canberra, ACT, Australia, in 2017 and 2018, respectively, where he is currently pursuing the Ph.D. degree in software engineering.

He was a Visiting Ph.D. Researcher with the University of Michigan, Ann Arbor, MI, USA, in 2020. His research interests include high-performance computing, bioinformatics optimization, and industry infrastructure software development.

Eric McCreath (Member, IEEE) received the B.E. and Ph.D. degrees in computer engineering from the University of New South Wales at Sydney, Sydney, NSW, Australia, in 1993 and 1999, respectively.

From 1999 to 2001, he was a Lecturer with the Department of Computer Science, The University of Sydney, Sydney. From 2001 to 2021, he was a Faculty Member with the Research School of Computer Science, Australian National University, Canberra, ACT, Australia, where he worked on employing novel architectures, such as GPUs and FPGAs, for computationally intensive tasks. He is currently a Senior Software Engineer with Skykraft Pty Ltd., Braddon, ACT, Australia.

Zhengya Zhang (Senior Member, IEEE) received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley (UC Berkeley), Berkeley, CA, USA, in 2005 and 2009, respectively.

He has been a Faculty Member with the University of Michigan, Ann Arbor, MI, USA, since 2009, where he is currently an Associate Professor with the Department of Electrical Engineering and Computer Science. His research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing.

Dr. Zhang was a recipient of the David J. Sakrison Memorial Prize from UC Berkeley in 2009, the National Science Foundation CAREER Award in 2011, the Intel Early Career Faculty Award in 2013, and the University of Michigan of College of Engineering Neil Van Eenam Memorial Award in 2019. He has been serving on the Technical Program Committees of Symposium on VLSI Circuits and the IEEE Custom Integrated Circuits Conference (CICC) since 2018. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS from 2013 to 2015 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS from 2014 to 2015. He has been an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS since 2015.