IEEE Open Journal of the
Solid-State Circuits Society

Received 15 June 2022; revised 14 August 2022 and 26 September 2022; accepted 13 October 2022. Date of publication 25 October 2022; date of current version 13 December 2022.

Digital Object Identifier 10.1109/OJSSCS.2022.3216798

A 64-TOPS Energy-Efficient Tensor Accelerator in 14 nm With Reconfigurable Fetch Network and Processing Fusion for Maximal Data Reuse

SANG MIN LEE[1] (Senior Member, IEEE), HANJOON KIM[1], JESEUNG YEON[1], JUYUN LEE[1], YOUNGGEUN CHOI[1], MINHO KIM[1], CHANGJAE PARK[1], KISEOK JANG[1], YOUNGSIK KIM[1], YONGSEUNG KIM[1], CHANGMAN LEE[1], HYUCK HAN[1], WON EUNG KIM[1], RUI TANG[1,2] (Senior Member, IEEE), AND JOON HO BAEK[1] (Member, IEEE)

[1]Hardware Department, FuriosaAI, Inc., Seoul 06036, Republic of Korea

[2]Marketing, Strategy and Operations Department, MSQUARE Ltd., Shanghai 201107, China

CORRESPONDING AUTHOR: S. M. LEE (e-mail: sage.lee@furiosa.ai)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government (MSIT, Development of Artificial Intelligence Deep Learning Processor Technology for Complex Transaction Processing Server) under Grant 2020-0-01309-001.

ABSTRACT For energy-efficient accelerators in data centers that leverage advances in the performance and energy efficiency of recent algorithms, flexible architectures are critical to support state-of-the-art algorithms for various deep learning tasks. Due to the matrix multiplication units at the core of tensor operations, most recent programmable architectures lack flexibility for layers with diminished dimensions, especially for inferences where a large batch axis is rarely allowed. In addition, exploiting the data reuse inherent within tensor operations for computing a single matrix multiplication is challenging. In this work, an extension of a vector processor in 14 nm is proposed, which is customized to tensor operations. The flexible architecture enables a tensorized loop to support various data layouts and different shapes and sizes of tensor operations. It also exploits all possible data reuse, including input, weight, and output. Based on the tensorized loop, fetch and reduction networks, which unicast or multicast with the ordering of both input data and processing data, can be simplified using a circuit-switching-like network with configured topology and flow control for each tensor operation. Two processing elements can be fused to optimize latency for a large model or can operate individually for throughput. As a result, various state-of-the-art models can be processed efficiently with straightforward compiler optimization, and the highest energy efficiency of 13.4 Inferences/s/W on EfficientNetV2-S is demonstrated.

INDEX TERMS AI accelerators, convolutional neural networks, data reuse, depth-wise and group convolution, inference, ML accelerator, ML processor, reconfigurable systems.

I. INTRODUCTION

MANY studies have considered how to improve the energy efficiency of deep learning accelerators. Supporting state-of-the-art deep learning algorithms is crucial to increasing this energy efficiency, primarily because deep learning algorithms are rapidly evolving to renew state-of-the-art accuracy every few months, and the new state-of-the-art algorithms usually achieve higher accuracy with fewer parameters and computations. Therefore, an architecture specialized for a specific model may achieve high energy efficiency for that model, but it does not show superior accuracy for a task and does not reflect the energy efficiency developed in a new algorithm. For example, EfficientNetV2 [1], one of the most efficient state-of-the-art models, demonstrates an accuracy that cannot be achieved with ResNet. If EfficientNetV2-B0 is compared

with ResNet-152, which shows similar accuracy, the parameter size is eight times smaller. Therefore, to support the present and near future state-of-the-art models, the accelerator should be sufficiently flexible to efficiently support a wide range of deep neural network (DNN) layers [2].

Flexibility affects not only energy efficiency from algorithmic advances but also affects transistor-level and system-wide energy efficiency. Because chips that support a wide range of deep learning tasks cover many applications, they can justify high development costs, enabling the development of more energy-efficient chips that use more advanced process nodes. In addition, supporting different tasks in data centers means that many users can share hardware resources, thereby increasing overall resource utilization.

To design an architecture that can accommodate various shapes and sizes of DNN layers, the data must be flexibly mapped spatially according to the specific shape and size of the layer rather than mapped to a set of preselected dimensions [2]. In a broader sense, flexible architectures must support various types of data reuse inherent in tensor operations. Since data movement consumes more energy than computation [3], [4] and often dominates the overall execution time in actual model execution [5], it is important to minimize memory access and to maximize data reuse near the computational unit to channel all given power to computation [6]. In addition, with only single data reuse, operands with low or no reuse start to dominate energy consumption, and overall energy savings can be limited [7].

Although many different specialized accelerators have been proposed [8], [9], [10], only a few have provided instruction set architectures (ISAs) and their programming models to support various tasks, and most of their ISAs have matrix multiplication instructions at the core of processing tensor operations. The matrix multiplication unit, which is often implemented as a systolic array [11], can utilize $N \times N$ computations with $\mathcal{O}(N)$ bandwidth and support horizontal and vertical data reuse, which enables more efficient computation compared to vector instructions. This is very suitable when most tensor operations can be partitioned into multiple matrix multiplications for deep learning, especially for training workload where a large batch size can be easily exploited as one of the reuse axes. However, as inference has different demands and often requires low latency, only a single or small batch size inference is allowed [12], [13]. As a result, energy-efficient and compact layers, which have diminished data dimensions, may not fully utilize the whole matrix multiplication unit [2]. Data reuse across the matrix multiplication boundary is also difficult. Tensor operations usually have more dimensions than those used by single matrix multiplication and have data reuse patterns that cannot be covered by matrix operations.

To address these challenges, we propose a tensor processor architecture, an extension of a vector processor [14], [15], [16], [17], [18] that is customized for tensor operations used in deep learning inference. A vector processor provides vector instructions along with scalar instructions. The vector instruction specifies operand vectors and the vector length so that each operation can be applied to span the full vector length. For example, if a vector C is calculated by adding two vectors A and B, then this vector operation is expressed as follows:

$$C_i = A_i + B_i, \quad i = 0, \ldots, V_L - 1$$

where V_L is the vector length.

Implementing data parallelism and pipelining is relatively easy in the vector processor owing to the independence of vectors; instruction fetch and decode overhead can be amortized because one instruction is executed V_L times. In addition, since memory accesses are in regular patterns, the maximum bandwidth of multiple memory banks can be utilized, and prefetching is straightforward.

However, the vector processor is efficient only when the data parallelism is regular, which is the case for the workloads of deep learning since they are given from frameworks that handle predefined tensor operations. Additionally, the compiler can maximize the memory bandwidth by lowering tensors for optimal bank access.

Since dot product operations occupy most of the operations in deep learning, dot product pipelines can be provided in addition to vector processing pipelines. In particular, by allowing the stream of tensors to be reused for multiple dot product operations, not only energy efficiency is improved but also the compute/memory operation balance can be increased by the number of data reuse times.

Moreover, since DNN computations are more specialized in tensor operations with N dimensions, we can further optimize such an architecture to fit deep learning inference. A tensorized loop rather than a vectorized loop is utilized with N dimension length and stride, and arithmetic units can operate on multiple accumulators and register files according to streaming data to exploit all possible data reuse patterns while removing unnecessary complexity. The main contributions of this work are as follows.

1) A metamorphic tensor processor is proposed that supports a tensorized loop and exploits all possible data reuse, including input, weight, and output. It has a flexible MAC datapath to support various data reuse patterns and to support various data layouts, such as NHWC or NCHW, with configurable temporal and spatial accumulation units, as different tensor operations have different optimal data layouts. This enables efficient support of a group or depth-wise convolution on top of the commonly used channel direction only. Many architectures do not support depth-wise convolution efficiently resulting in significant execution time increase [19], [20].
2) This processor also has a transpose engine and a vector engine with N-dimension indexing to support tensor manipulations and the various vector operations required by deep learning models.
3) ISA provides an asynchronous load and store that enables the full utilization of computation units while

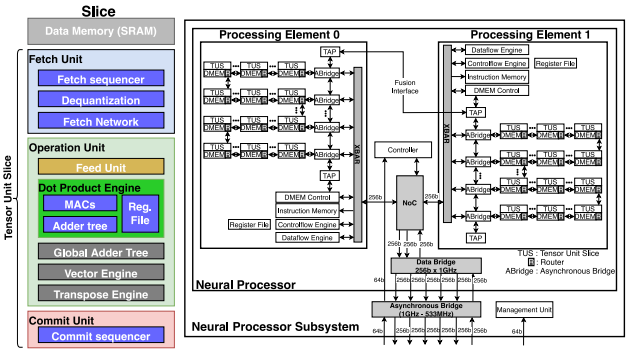


FIGURE 1. Slice with a TUS and the neural processor.

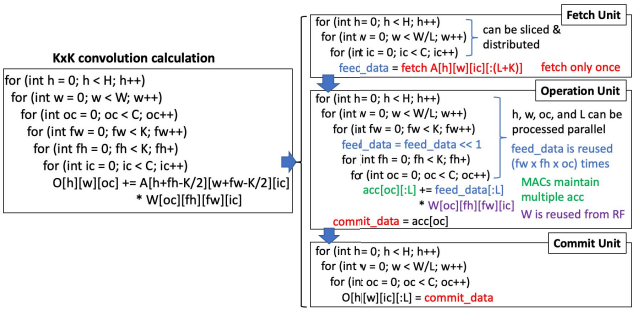


FIGURE 2. Optimal data flow for the $K \times K$ convolution.

hiding memory latency for a large model, which needs to be stored in external DRAMs.

4) A single processing element (PE) has 64 slices, which comprise an SRAM and a compute pipeline. Whole slices can be configured as a single unit that can utilize all of its MACs using a single tensorized loop, simplifying communication among slices and easing compiler optimization compared to hundreds or thousands of cores that require its own individual instruction stream.
5) Two PEs in the neural processing unit (NPU) can operate separately or as one unit in fused mode to support the optimal number of slices that could differ depending on the model. In particular, when the model is large, the fused mode is more efficient compared with the use of separate PEs by reducing memory usage and communication overhead. Our fetch and reduce networks among the slices allow multiple PEs to be fused easily with a simple datapath connection.

The proposed processor is flexible enough to support extensive tensor operators such that it supports more than 30 models, including FBNet [21], MobileNet [22], ResNet [23], SSD [24], EDSR [25], and even the state-of-the-art EfficientNetV2 [1], on various tasks, such as image classification, object detection, and super resolution. In addition, this work demonstrates a much smaller chip area and power consumption compared to edge-oriented accelerators, despite having the same data center performance level.

II. ARCHITECTURE OVERVIEW

The NPU comprises two PEs (Fig. 1). Each PE can process instructions individually with a Controlflow engine, which is a custom-developed in-order core with a custom ISA. In particular, the ISA has a tensor ISA (T-ISA) that controls data memory (DMEM) and a tensor unit (TU). Tensors processed by TU can be sliced and stored in DMEM as an optimal data layout to be distributed and processed by slices of TU. Each TU slice (TUS) includes a DMEM and computes pipeline, a TU is composed of 64 slices.

A. TENSOR PIPELINE

The T-ISA is an extension of a vector processor [15] used to process tensors, and it has fetch, operation, and commit pipelines for N-dimensional tensors. When these pipelines are configured, the whole pipeline consecutively executes computation on the entire tensor operator. Configuration refers to the indexing structure of the fetch, operation, and commit units. For example, the fetch unit generates an address toward what the fetch sequencer will fetch. The fetch sequencer can generate indexing through, but is not limited to, height, width, and input channels. Such an indexing structure is specialized for tensor operators but is flexible enough for most tensor indexing patterns. Each slice performs the aforementioned pipeline, which is individually controlled by the same special function register (SFR) configuration. The Controlflow engine core writes or broadcasts to SFRs of slices.

For example, let us consider a $K \times K$ convolution algorithm. H, W, and C represent the tensor's height, width, and channel direction size, respectively [6]. Fig. 2 illustrates which datapath would be the most efficient considering how it may be formed from the algorithm perspective. This can be considered as dividing the pipeline into fetching data from SRAM, running the operation, and then committing the data. To compute the output, input activation and weight are repetitively utilized. Ideally, the fetch unit will fetch input activation only once from the SRAM. The fetched data will be fed to the operation unit.

The operation unit can reuse the feed data to the extent of the multiplied results of the filter width, filter height, and output channel. For reusability in the filter width direction, the feed data can be shifted. To reuse one feed data multiple times, the partial sum must be maintained by using many accumulators on the various outputs that use the feed data. As explained earlier, maximizing the data reuse will minimize the power of data movement and latency. Depending on the shape of the tensor and target operation, various reuses of input, weight, and output are possible. The data that completed the accumulation needs to be sequentially delivered to the commit unit and stored in designated locations. The accumulator-related units will be explained later. Depending on the tensor size, a TU can process millions

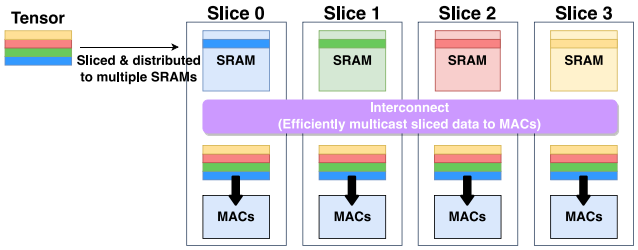


FIGURE 3. Distributed MACs controlled as a single unit with tensors partitioned, distributed, and stored into multiple slices.

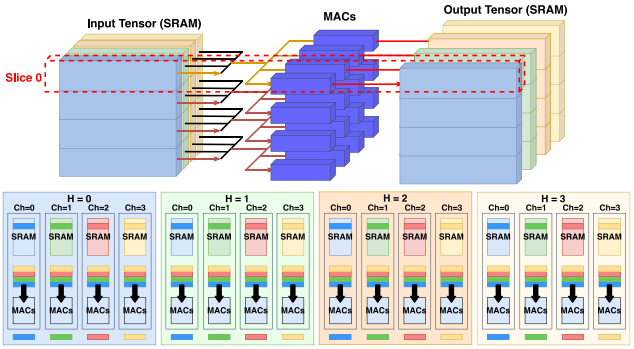


FIGURE 4. Distributed point-wise convolution through 16 slices with 4 in the height direction and 4 in the channel direction.

of operations per single SFR configuration. Supported by a fetch sequencer that generates addresses, the fetch unit provides complex memory access and computations, such as padding and de-quantization.

B. FETCH NETWORK

Slices are interconnected with a fetch network (FN). Ideally, the memory near the slice could minimize idle cycles for optimal performance, but this limits tensor operations since the same data may need to be stored repeatedly. In this work, tensors can be partitioned, distributed, and stored into multiple data memory slices (Fig. 3), and the FN can multicast the same data into multiple slices simultaneously during the execution stage to reuse the data in the MACs. While the data are actually sent in packet units, they are sent in consistent and preconfigured source-to-destination routes, which is comparable to circuit switching as a fixed topology while each packet has a destination address and is routed and flow-controlled independently in conventional packet switched networks. A circuit-switched FN assumes that each source always delivers data to the same destination or destinations while one tensor operation is in progress. In other words, during one operation, each source is connected to the designated destinations, and the FN router for the connections is preconfigured on how to route and flow control. In addition, data ordering from multiple sources is maintained. Hence, the operation unit can sequentially process the fetched data straightforwardly. Moreover, as the entire tensor operators are delivered in a consistent pattern, almost the full bandwidth can be utilized, whereas simple flow control is used with lightweight routers without a virtual channel. In the measurement, predictable latency and throughput are observed as expected. In other words, the distributed MACs are controlled as single units rather than as individual cores. Through these processes, no replication in SRAM is required. For each slice, the DMEM size is 0.243 mm^2 and the FN occupies 0.012 mm^2, which is only 4.95%. Therefore, the area and power saving is significant since SRAM does not have to store copies of required data locally. In addition, during the overall tensor operation, the data are transferred in a simplified and efficient manner in an established configuration. As a comparison, a conventional packet-switched network incurs a large variation in latency. To mitigate or hide unpredictable latency, large buffers or thread contexts are needed. Another approach is the static compilation of the network, which induces a complex problem for the compiler to resolve [26].

Fig. 4 demonstrates how the input tensor passes through the MAC and its output tensor. The input tensor is sliced in the directions of channel and height. The diagram shows how 16 slices are stored, with four tensors sliced in the height direction, and four tensors sliced in the channel direction. The MACs that compute the respective output channels from input channels by receiving four sets of data receive the aforementioned data by multicast and execute computation in a parallel manner. The computation results are again stored in the slice SRAM. In another example, the FN can be configured into multiple rings while executing one tensor operator [Fig. 5-1.(a)]. The pattern in Fig. 5-1.(c) supports the depth-wise separable or group convolution such that the highest distributed SRAM bandwidth utilization is expected for these operations, and conventional convolutions are also supported [Fig. 5-1.(a) and (b)].

C. OPERATION UNIT

Data are transferred to the operation unit after being fetched by the FN. The operation unit consists of a dot product engine (DPE) that includes multiple INT8 MACs, a feed unit (FU), a vector engine, and a transpose engine, as shown in Fig. 6. The DPE conducts the dot product computation, which takes up most of the neural network computations. There are 16K of MAC units in total. An FU can supply the same data to a DPE multiple times, with the capability of shifting the data left or right with various strides. Depth-wise separable convolution often has 10× lower compute density than regular 2-D convolution, and the FU enables depth-wise convolution with data reuse of both the width and height. This flexibility enables data reuse of the kernel size, which then enhances energy efficiency and performance in a given SRAM bandwidth. Multiple accumulators set as output stationary in a DPE temporally accumulate into multiple output channels or rows by reusing input data. Accumulators support nonlinear computations using the feed data as table indices. Table entries are precalculated and stored in the accumulators' registers. A DPE can perform a partial sum

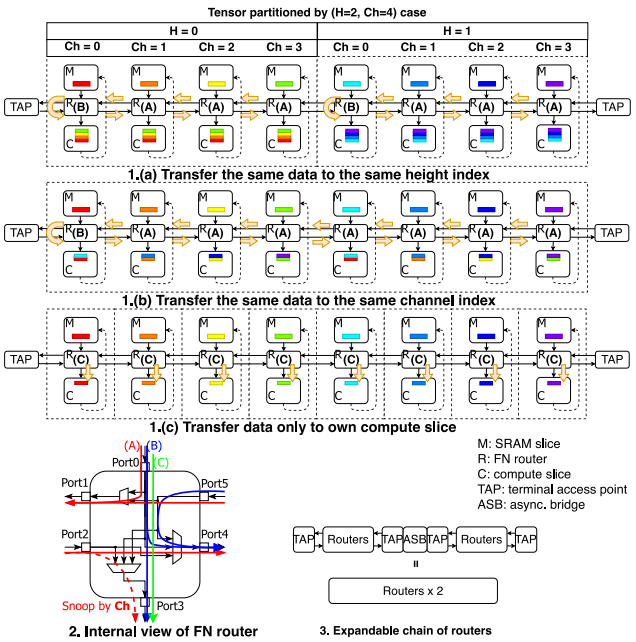


FIGURE 5. FN and operation examples for transferring the 1.(a) same data to the same height index configured into multiple rings, 1.(b) same data to the same channel index, and 1.(c) data only to its own slice for depth-wise separable convolution.

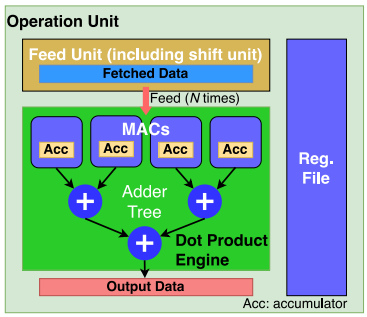


FIGURE 6. DPE with multiple MACs, an FU, and a local adder tree.

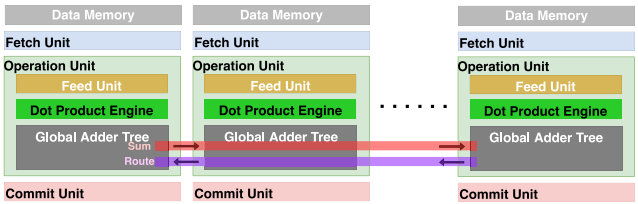


FIGURE 7. GAT with the partial sum and output paths distributed across multiple slices.

spatially depending on the tensor shape and operation using an adder tree. The compiler can find optimal temporal and spatial configurations and suitable memory layouts of tensors on DMEM, such as NCHW or NHWC. Weights or activations can be preloaded into the tensor register file and reused during the entire tensor operation. The DPE can also provide a spatial sum using a depth-configurable adder tree. By providing both temporal and spatial accumulations, all computations can be supported, including width-last and channel-last, according to the tensor operator. In depth-wise convolution, the width-last can be used, which reuses data in the width direction, thereby utilizing the compute unit as much as possible within the given SRAM bandwidth. For a stride of N, we can set the adder tree depth as $\log N$ and achieve efficient computation. In the case of single stationary data reuse from input, weight, and activation reuse, energy consumption from nonstationary data increases substantially [7]. To resolve this issue, this work is designed to provide all types of data reuse, including input, weight, and activation. In instances where all the computation cannot be done in one slice or where computation is divided across several slices, the DPE output may pass through the global adder tree (GAT), which is distributed among slices and has configurable depth. We can execute a reduction of the partial sum or finding of the maximum computed by several slices through the GAT, and the output can be routed to a desired slice, as shown in Fig. 7. The GAT is connected in a ring topology as the FN. Since the GAT is pipelined, its latency is hidden by a pipeline that streams entire tensors which is similar to other units' latencies. Therefore, all types of communication patterns are supported through FN and GAT. FN enables unicast and multicast both weights and activations, whereas GAT enables unicast partial sums and activations, thereby supporting all types of communication patterns.

The data that have undergone dot product computation, if necessary, may selectively pass through the vector computation pipeline. The vector engine, even without DPE, along with the fetch unit and commit unit, can be used to perform standalone pipelined computations, such as ReLU, vector addition, subtraction, multiplication, shift, and clip. The vector engine resembles a vector processor in many ways. For example, vector processors, such as RISC-V RVV or ARM SVE, can pipeline vectors by setting the vector length. Similarly, the vector engine is configured by SFRs, processing data through a pipeline. However, the difference is that vector processors process 1-D variable vector length, whereas the proposed architecture supports N-dimensional variable length tensors. In addition, the proposed architecture processes vector operations in a parallel manner across multiple slices, according to the configurations. In other words, TU could in itself be seen as an extension of the vector processor specialized for N-dimension tensors and dot products.

The computation results should be reshapable so that they can be easily used for the next computation. Depending on the tensor operator, some operators are optimal in channel-last, whereas others are optimal in width-last. Hence, tensors must be transposable in relation to subsequent computations. Various tensor manipulations, such as split, concatenation, and reshaping, may also be needed. The transpose engine supports the transpose for the last axis of the tensors. The commit unit can, among others, transpose, split, and concatenate tensors, except for transposing the last axis of tensors. The transpose engine and commit unit can

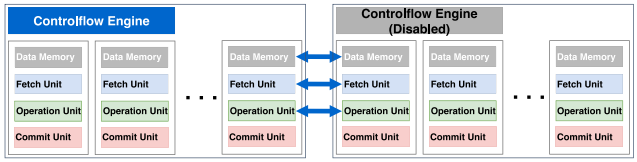


FIGURE 8. PE fusion.

process the aforementioned computation results on-the-fly, so a separate operator for tensor manipulation is not needed.

In summary, communication among multiple slices and the pipeline within each slice are described. Regarding this overall pipeline, the objectives are as follows.

1) Minimizing SRAM usage.
2) Minimizing SRAM access to one time only.
3) Efficiently multicast only to neighbor slices for parallel computation.
4) Exploiting every data reuse within the DPE.
5) Supporting any axis type in tensor operation to enhance computation efficiency.
6) A single configuration controls whole tensor operations via SFRs, thereby minimizing the control overhead.
7) The architecture must be flexible enough to support most indexing patterns in tensor operations and tensor manipulations.

D. PROCESSING ELEMENT FUSION

One PE has a TU comprising 64 slices. This equates to a peak performance of 32 TOPS and 16-MB SRAM per TU. Two PEs in the NPU can operate separately or as one unit in the fused mode when the control, FN, and GAT are connected, as shown in Fig. 8. In the fusion mode, one core controls the fused TU entirely, which comprises 128 slices, as if they were one PE. The routing ID is extended by tagging one bit of the PE ID to the slice ID. For some models, even if all 128 slices are utilized, the performance does not scale linearly. However, for some other models, the performance is enhanced in a superlinear fashion with more slices. This is because when a model is large, the fused PEs can reduce memory usage by reducing the replication of weights or activations and, in turn, reduce the communication overhead, programming complexity, and execution latency. In other words, fused PEs can handle tensor parallelism natively within a chip for large models.

III. IMPLEMENTATION AND MEASUREMENTS

The chip comprises four subsystem blocks, as depicted in Fig. 9.

The CPU subsystem consists of quad RISC-V U74 cores [27]. Each core has 32 kB of L1 I-Cache, 32 kB of L1 D-Cache, and 2 MB of L2-cache shared among the core complexes. The maximum operation frequency for each core is 1.0 GHz. The CPU core can perform 4-channel platform DMA (PDMA) and preprocessing or postprocessing for some

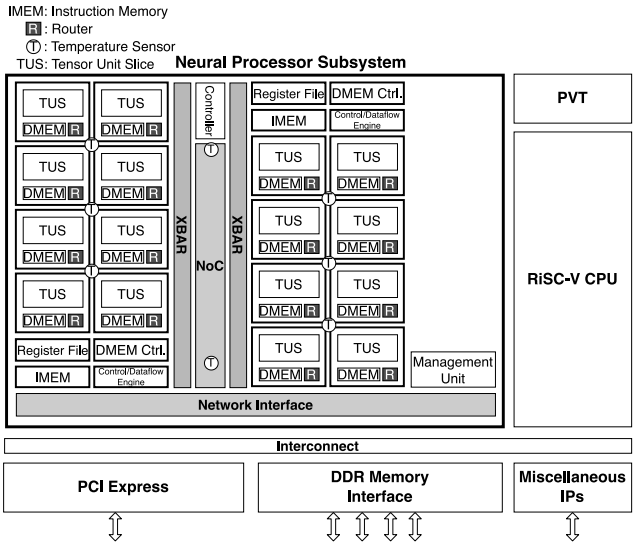


FIGURE 9. Top-level block diagram showing various subsystems: CPU, PCIe, memory, peripherals, and neural processor.

specific models. It also manages and maintains the chip under various external server conditions. It reads the on-chip temperature sensors periodically and performs dynamic voltage and frequency scaling (DVFS) accordingly by controlling PMIC voltages through GPIO and by controlling internal PLLs. Various peripherals are also integrated, and PWM controls fan speed.

The PCIe subsystem has a PCIe Gen4 controller and a PHY with eight lanes. The PCIe internal DMA is used for large data transfer between the host DRAM and the device DRAM, and it supports a bandwidth of 16 GB/s.

The memory subsystem has four LPDDR4x memory channels. The width of each channel is 32-DQ with 4266 MHz, meaning that each channel supports 17 GB/s, and the four channels provide 67 GB/s of aggregated bandwidth in an interleaved mode. For efficient bandwidth utilization, the internal interconnect bus provides memory channel interleaving. Each application can choose 1-channel (no interleaving), 2-channel, or 4-channel interleaving. The interconnect bus is implemented with a TileLink bus with TileLink protocols.

The neural engine subsystem comprises two PEs and a neural engine controller (NEC) to control the PEs. NEC has two dedicated PLLs, glitch-free clock dividers, and multiplexers such that the clock can be adjusted according to the DVFS table based on the workload and temperature. The dynamic clock change also stabilizes the power supply rails. For example, if a heavy workload is executed, multiple MACs operate concurrently and incur large in-rush currents. The large current, in turn, can cause a large voltage drop on the power supply rails. Therefore, the warm-up period is supported to prevent this by incrementally increasing the clock with a programmable warm-up time.

As each PE has 8K of MAC units and 16 MB of data memory, it is quite challenging to operate all synchronously

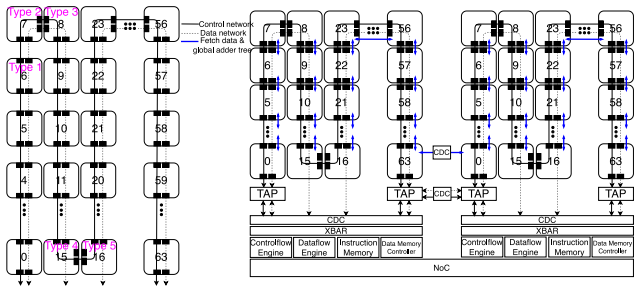


FIGURE 10. Five types of slices placed as an 8×8 matrix in a ring topology and the fusion path with clock domain crossings (CDCs) and terminal access points (TAPs).

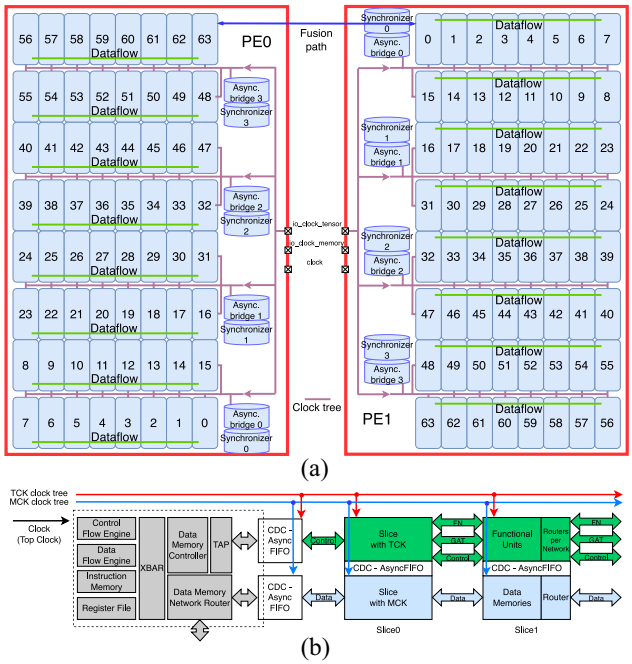


FIGURE 11. PE. (a) Floor plan of 64 slices per PE and (b) clock distribution to operate the TU synchronously.

at 2.0 GHz. Operating the MAC and memory simultaneously can also cause large voltage drops on power rails, so PE is implemented using multiple TUSs. The performance and TUS size need to be decided upon considering the tradeoff between the physical implementation feasibility for the large size and the communication overhead for the small size. As a result, 64 slices per PE are implemented, and the gate count of one slice is approximately 4M to ensure that the place-and-route iteration time is reasonable. However, as mentioned previously, dividing the PE into multiple slices requires data transfer between them, and the entire PE is designed to be synchronous so that the complexity of the interconnect network routers is reduced and PE execution is predictable cycle-accurately.

The aforementioned two dedicated PLLs drive a tensor clock (TCK) of 2.0 GHz and a memory clock (MCK) of 1.2 GHz, and TCK drives the logic and MCK drives the memory. Because they are generated from two individual PLLs, clock frequency selection is more flexible. TCK and MCK are fed into slices with separate clock trees and are individually synchronous across all slices in the TU but are asynchronous inside the slice, which reduces the implementation overhead, while the communication path between the slices is synchronous for the deterministic communication latency. Slices inside the TU are connected by networks that can support up to 1 TB/s. All networks are in a ring topology, and each ring size is configured according to the network type. The ring topology is chosen since it is efficient in multicast while keeping data ordering to the destination nodes. This enables predictable data feeding to the destination with high throughput for the traffic pattern of the FN. Sixty-four slices are tiled as an 8×8 matrix, as depicted in Fig. 10. For short connections without detour and minimum latency by abutting slices and for simplicity in place-and-route, five types of slice macros are generated, even though their functions are the same. The internal memory is distributed among the slices, and the total size is 32 MB.

A clock spine structure is implemented to synchronously operate the TU, which has a size of approximately 50 mm^2 as shown in Fig. 11(a), which enables flexible arrangement and an interconnected topology. The purple line depicts both TCK and MCK. Internally, there are asynchronous FIFOs between the TCK and MCK domains, such that the two domains are asynchronous, as shown in Fig. 11(b). However, the connections between slices are synchronous. FN, GAT, and control networks are all synchronous to TCK, whereas the data network is synchronous to MCK. 64 slices are connected with synchronous TCK and synchronous MCK individually, as explained above. Slices are connected through each side only owing to ring topology interconnection, so the clock skew is minimized by maximally extending the common clock path. For example, meeting the timing requirements between the source registers of slice 0 and the destination registers of slice 1 is easy since clock ports are near the IO for both slices, and that between slice 7 and slice 8 is also easy since the relative clock network delay is small while most of the clock network is common.

The nominal core voltage and frequency are 0.8 V and 2.0 GHz, respectively. The voltage and frequency can be chosen dynamically according to the workload to optimize power and performance. The software controls thermal throttling with internal temperature sensors to prevent excessive junction temperatures. As shown in Fig. 12, the constant junction temperature (T_j) of approximately 104 °C is measured in a limited cooling environment while the workload is executed approximately starting at 70 s and ending at 2100 s. The average power consumption of PEs in this interval is 29 W. There are 15 temperature sensors across the chip, including the PE core, DRAM controllers, PCIe controllers, and CPU. The average is used to control the frequency and voltage in the 1-s interval. In Fig. 12, the voltage used is

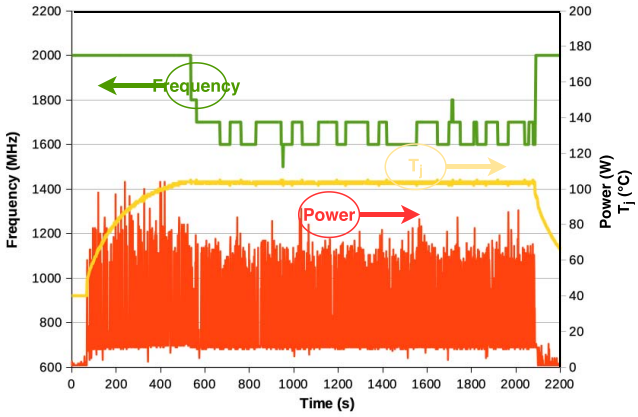


FIGURE 12. DVFS.

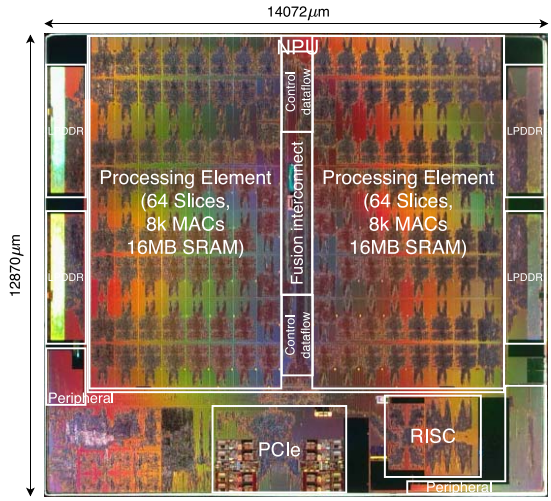


FIGURE 13. Die micrograph.

0.875 V, except at 1.5 GHz of core frequency when a lower 0.850 V is used. The 181-mm^2 chip is implemented in 14-nm FinFET CMOS technology, as shown in Fig. 13.

Since the NPU supports INT8, the FP models need to be converted to the INT8 models by quantization-aware training or post-training quantization in TensorFlow or PyTorch. The custom compiler supports models exported in TFLite and ONNX, and the generated binary can be executed by Python, C++, or Rust API, similar to the conventional execution flow. To generate the final binary that can compute the model in the NPU, multiple steps of intermediate representation (IR) are generated, and each IR requires various optimization steps, such as network partitioning, memory optimization, memory prefetch, and memory layout. A custom runtime library loads binary into instruction memories in PE once. For every inference, runtime transfers only input data into DRAM by DMA. The inference is then executed with input data loaded into DMEMs. NPU can improve energy efficiency by pinning weights into DMEMs according to model size. Multiple PEs can perform inferences independently while sending and receiving data asynchronously.

TABLE 1. Measurement comparison of ResNet50 per layer.

Layer	Nvidia, NPU [28]		This work	
	Latency (μs)	Energy (μJ)	Latency (μs)	Energy (μJ)
conv1	41.00	1050.60	19	890.18
pool1			5	73.21
res2a_branch1	8.87	241.02	7	186.98
res2a_branch2a	6.44	164.47	3	70.42
res2[a-c]_branch2b	9.26	284.24	10	319.49
res2[a-c]_branch2c	8.87	241.02	7	186.98
res2[b-c]_branch2a	14.04	468.24	6	180.17

Therefore, the compiler can use data or model parallelism according to the mode size and the required computation.

The architectural advantage of the proposed work is demonstrated by measurements of some of the first layers of ResNet50 [23] as shown in Table 1. For the first step of power consumption measurement per layer for a deep learning model, the task was split at the compiler level for each layer as a unit. Then, each layer was continuously iterated for approximately 10 000 times. Latency was calculated by averaging profiling results. At the same time, the average current and voltage of NPU were measured at the power source, and then unit power consumption per layer was calculated. Finally, layer-by-layer energy was calculated as a multiplication of power consumption with previously calculated latency. The results are better or comparable to the performance of the previous work [28].

INT8 performances are measured using various models, such as EfficientNetV1 B4, ResNet18, ResNet50, SSD-MobileNet, SSD-ResNet, and EfficientNetV2, which is a state-of-the-art model, to demonstrate flexibility. ImageNet datasets and post-training quantization are used. Mainly the edge-oriented processors with data center-level performance and the standard quantization scheme in TensorFlow and PyTorch are compared without pruning, as shown in Table 2. Calibration and measurements are based on inference rules for the closed division of MLPerf [29], which limits the lower bound of the quality to 99% of FP32. Before processing by the NPU, input images are resized to 224 $\times$ 224 $\times$ 3 and then lowered to a 256 $\times$ 256 $\times$ 4 tensor with padding by the host CPU. The performances are measured with a custom-designed PCIe card that is plugged into a server, as shown in Fig. 14. The total power of the card is measured, including DRAM and even voltage conversion losses. Each PCIe card has one chip mounted with a heat sink, and six cards are plugged into a server. The results for this work are measured at 0.8 V and 2.0 GHz.

Fig. 15 shows the power breakdown of a TUS by simulation and efficiencies on EfficientNetV2-S across core voltages by the measurements where the frequency has been swept in 100 MHz steps. Almost half of the power is consumed by the MAC, and register files consume 30.1%. The NPU can operate between 0.75 and 0.90 V with a maximum core frequency of 1.3–2.2 GHz accordingly.

In Table 2, system-level metrics of Inferences/s and Inferences/s/W are included on top of the low-level metrics, such as TOPS and TDP, to assess the software

TABLE 2. Performance summary and comparison to prior works.

		Xavier, Nvidia [31]	T4, Nvidia [32]	A2, Nvidia [33]	A30, Nvidia [34]	CHIMERA [38]	HanGuang, Alibaba [39]	**This work**
Performance		32 TOPS (INT8) 16 TFLOPS (FP16)	130 TOPS (INT8) 65 TFLOPS (FP16/FP32)	36/72 TOPS (INT8/4) 18 TFLOPS (FP16/BF16) 9 TFLOPS (TF32)	330/661 TOPS (INT8/4) 165 TFLOPS(FP16/BF16) 82 TFLOPS (TF32)	0.92 TOPS (INT8/FP16)	825 TOPS (INT8)	64, max. 70.4 TOPS (INT8)
Process (nm)		12	12	8	7	40	12	14
Area (mm^2)		350	545	N/A	826	29.2	709	181
Clock (MHz)		1377	1590 (boost)	1770 (boost)	1440 (boost)	200	700	2000 (nom.) (1300–2200)
PCIe		Gen4 ×8	Gen3 ×16	Gen4 ×8	Gen4 ×16	N/A	Gen4 ×16	Gen4 ×8
On-chip SRAM (MB)		9.5+	18	N/A	N/A	0.5/2.2 (SRAM/RRAM)	192	32
External DRAM	Type	LPDDR4x, 4266	GDDR6	GDDR6	HBM2			LPDDR4x, 4266
	BW (GB/s)	137	320+	200	933	N/A	N/A	66
	Size (GB)	16	16	16	24			32
TDP (W)		30	70	40-60	165	0.126	280	60
Latency† (ms)	EfficientNetV2-S	7.7 [35]	4.57§	2.88§	2.85§			**1.70**§
	EfficientNetV1 B4	8.6 [35]						**2.97**
	ResNet18					60.0		**0.296**
	ResNet50	1.98 [36]	0.85 [36]	0.71 [37]	0.52 [37]		0.17 [36]	**0.71** [37]
	SSD-MobileNet	0.58 [36]		0.45 [37]	0.33 [37]			**0.36** [37]
	SSD-ResNet			14.64 [37]	3.03 [37]			**13.43** [37]
Throughput† (Inferences/s)	EfficientNetV2-S	129.9	218.8	347.2	350.9			**588.2**
	EfficientNetV1 B4	116.3						**336.7**
	ResNet18					16.7		**3378.4**
	ResNet50	505.1	1176.5	1408.5	1923.1		5882.4	**1408.5**
	SSD-MobileNet	1724.1		2222.2	3030.3			**2777.8**
	SSD-ResNet			68.3	330.0			**74.5**
MAC utilization† (%)	EfficientNetV2-S	4.3	1.8	10.2	1.1			**9.8**
	EfficientNetV1 B4	1.5						**2.2**
	ResNet18					6.6		**19.2**
	ResNet50	12.9	7.4	32.0	4.8		5.8	**18.0**
	SSD-MobileNet	13.3		15.3	2.3			**10.7**
	SSD-ResNet			82.1	43.3			**50.4**
Energy efficiency† (Inferences/s/W)	EfficientNetV2-S	4.33	3.13	5.79	2.13			**13.4**
	ResNet18					132.3		**128.1**
	ResNet50	16.84	16.81	23.48	11.66		21.01	**27.84**
	SSD-MobileNet	57.47		37.04	18.37			**71.96**

†Latency, throughput, MAC utilization, and energy efficiency are with the single batch size and INT8.
§Size 300×300, CUDA 11.6, ONNX model is executed by trtexec in TensorRT 8.4, and conditions are the same across all accelerators.

FIGURE 14. Six PCIe cards mounted with the chips and passive heat sinks are plugged into a server.

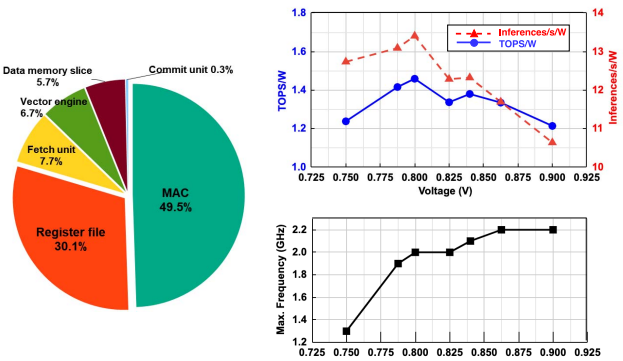


FIGURE 15. TUS power breakdown and efficiencies on EfficientNetV2-S and the maximum core frequency across the core voltage sweep. TOPS/W in the figure is peak architectural TOPS/W.

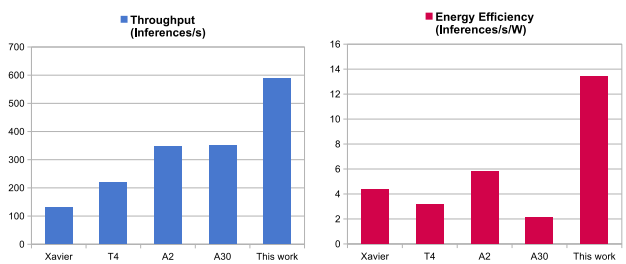


FIGURE 16. Performance comparison to prior works on EfficientNetV2-S.

and system aptitude of the processor [30]. Latency and power are measured with a single batch, which is important for the data center [12]. The MAC utilization is the highest except A2 and SSD-MobileNet of Xavier. The energy efficiency of EfficientNetV2-S, ResNet50, and SSD-MobileNet is 13.4, 27.84, and 71.96 Inferences/s/W, respectively; this is the highest among the comparisons. The average power consumption for each case was 43.9, 50.6, and 38.6 W. As shown in Fig. 16, the energy efficiency of EfficientNetV2-S is 2.31× higher than Nvidia A2, and 6.29× higher than Nvidia A30 in INT8. The latency and throughput of EfficientNetV2-S are the best with 1.70 ms and 588.2 Inferences/s, respectively. The energy efficiency of 128.1 Inferences/s/W is comparable to the on-chip resistive RAM implementation of 132.3 Inferences/s/W [38], whereas the throughput is 202× higher.

IV. CONCLUSION

The proposed work shows that an extension of vector processors customized for tensor operations can maximize performance by exploiting every parallelism of models in a single or small batch. By maximizing data reuse, this work utilizes computation units, even for operators such as depth-wise convolutions that require high SRAM bandwidth. In addition, SRAM access and data movement are minimized, and the data reuse contained in tensor operators is maximized. In other words, the models can be run efficiently with high accuracy and fewer operations. The proposed architecture is flexible enough to support the extensive indexing patterns contained in tensor operators, given that it is specialized for tensor operators. The compilers simply need to define these indexing structures; thus, compilation is easy. Based on user models and needs, it is easy to provide data, pipeline, and tensor parallelism. A compiler-friendly architecture without sacrificing performance would be the most important aspect of the future NPU. This approach can be further scaled to higher computation power systems.

REFERENCES

[1] M. Tan and Q. V. Le, "EfficientNetV2: Smaller models and faster training," 2021, *arXiv:2104.00298*.

[2] 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 J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[3] R. Hameed et al., "Understanding sources of inefficiency in general-purpose chips," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 37–47, Jun. 2010.

[4] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[5] Y. Ju and J. Gu, "A 65nm systolic neural CPU processor for combined deep learning and general-purpose computing with 95% PE utilization, high data locality and enhanced end-to-end performance," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2022, pp. 248–249.

[6] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[7] R. Venkatesan et al., "MAGNet: A modular accelerator generator for neural networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Dec. 2019, pp. 1–8.

[8] S. Davidson et al., "The Celerity open-source 511-core RISC-V tiered accelerator fabric: Fast architectures and design methodologies for fast chips," *IEEE Micro*, vol. 38, no. 2, pp. 30–41, Mar./Apr. 2018.

[9] P. Vivet et al., "2.3 A 220GOPS 96-core processor with 6 chiplets 3D-stacked on an active interposer offering 0.6ns/mm latency, 3Tb/s/mm^2 inter-chiplet interconnects and 156mW/mm^2@ 82%-peak-efficiency DC-DC converters," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 46–48.

[10] F. Zaruba, F. Schuiki, and L. Benini, "Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing," *IEEE Micro*, vol. 41, no. 2, pp. 36–42, Mar./Apr. 2021.

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

[12] M. Anderson et al., "First generation inference accelerator deployment at Facebook," 2021, *arXiv:2107.04140*.

[13] Y. LeCun, "1.1 deep learning hardware: Past, present, and future," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 12–19.

[14] K. Asanović, "Vector microprocessors," Ph.D. dissertation, Electr. Eng. Comput. Sci., Dept., Univ. California, Berkeley, CA, USA, 1998.

[15] R. M. Russell, "The CRAY-1 computer system," *Commun. ACM*, vol. 2, no. 1, pp. 63–72, Jan. 1978.

[16] B. Zimmer et al., "A RISC-V vector processor with simultaneous-switching switched-capacitor DC–DC converters in 28 nm FDSOI," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 930–942, Apr. 2016.

[17] A. Gonzalez et al., "A 16mm^{2}106.1 GOPS/W heterogeneous RISC-V multi-core multi-accelerator SoC in low-power 22nm FinFET," in *Proc. IEEE ESSCIRC*, Sep. 2021, pp. 259–262.

[18] D. R. Ditzel, "Accelerating ML recommendation with over 1,000 RISC-V/tensor processors on Esperanto's ET-SoC-1 chip," *IEEE Micro*, vol. 42, no. 3, pp. 31–38, May/Jun. 2022.

[19] J. Park et al., "Deep learning inference in Facebook data centers: Characterization, performance optimizations and hardware implications," 2018, *arXiv:1811.09886*.

[20] D. Zhang et al., "A full-stack search technique for domain optimized deep learning accelerators," in *Proc. 27th ACM Int. Conf. Archit. Support Programming Lang. Oper. Syst.*, 2022, pp. 27–42.

[21] B. Wu et al., "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10726–10734.

[22] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. *arXiv:1704.04861*.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[24] W. Liu et al., "SSD: Single shot multibox detector,'in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2016, pp. 21–37.

[25] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 1132–1140.

[26] D. Abts et al., "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads," in *Proc. 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2020, pp. 145–158.

[27] "U74." SiFive. 2020. [Online]. Available: https://www.sifive.com/cores/u74

[28] B. Zimmer et al., "A 0.32-128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, Apr. 2020.

[29] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, Sept. 2020, pp. 446–459.

[30] G. W. Burr, S. Lim, B. Murmann, R. Venkatesan, and M. Verhelst, "Fair and comprehensive benchmarking of machine learning processing chips," *IEEE Design Test*, vol. 39, no. 3, pp. 18–27, Jun. 2022.

[31] M. Ditty, A. Karandikar, and D. Reed, "NVIDIA's Xavier SoC," in *Proc. IEEE Hot Chips 30 Symp. (HCS)*, 2018, p. 4.

[32] "NVIDIA turing GPU architecture." NVIDIA. Accessed: Sep. 14, 2018. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[33] "NVIDIA A2 tensor core GPU." NVIDIA. Accessed: Mar. 2022. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/solutions/data-center/a2/pdf/a2-datasheet.pdf

[34] "NVIDIA A30 tensor core GPU." NVIDIA. Accessed: Mar. 2022. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/data-center/products/a30-gpu/pdf/a30-datasheet.pdf

[35] "Imagenet (ILSVRC-2012-CLS) classification with EfficientNet V2 with input size 480x480." TensorFlow Hub. Accessed: Jul. 19, 2021. [Online]. Available: https://hub.tensorflow.google.cn/google/imagenet/efficientnet_v2_imagenet1k_m/classification/2

[36] "Machine learning innovation to benefit everyone, v0.5/v1.0." MLCommons. Accessed: Apr. 21, 2021. [Online]. Available: https://mlcommons.org/en/

[37] "V2.0 Results." MLCommons. Accessed: Apr. 6, 2022. [Online]. Available: https://mlcommons.org/en/inference-edge-20/

[38] M. Giordano et al., "CHIMERA: A 0.92 TOPS, 2.2 TOPS/W edge AI accelerator with 2 MByte on-chip foundry resistive RAM for efficient training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2021, pp. 1–2.

[39] Y. Jiao et al., "7.2 A 12nm programmable convolution-efficient neural-processing-unit chip achieving 825TOPS," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 136–140.

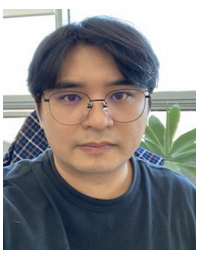
SANG MIN LEE (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Seoul National University, Seoul, South Korea, in 1997 and 1999, respectively, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2010.

In 2000, he joined Science and Engineering Services, Inc., Columbia, MD, USA, where he was a Senior Electrical Engineer in charge of analog and digital circuits for several LIDARs and mass spectrometers. From 2010 to 2017, he was with Qualcomm, San Diego, CA, USA, where he was involved in the design of data converters for cellular applications as a Senior Staff Engineer. From 2017 to 2020, he was with Samsung Electronics, Suwon, South Korea, where he designed circuits for 5G communications as a Principal Engineer. Since November 2020, he has been with FuriosaAI, Inc., Seoul, where he is a Vice President of Hardware Product Engineering. His current research interests include AI accelerator system and neuromorphic computing.

Dr. Lee is a recipient of the Bronze Medal from the 5th Human Tech Thesis Prize sponsored by Samsung Electronics.

HANJOON KIM received the B.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2005 and 2015, respectively.

He is currently a Co-Founder and the Chief Technology Officer of FuriosaAI, Inc., Seoul, South Korea, and leading AI chip architecture and development. His research interests include neural network accelerator architecture and interconnection network.



JESEUNG YEON received the B.S. degree in electrical and electronic engineering from Chung-Ang University, Seoul, South Korea, in 2009, and the M.S. degree in electrical engineering from the Pohang University of Science and Technology, Pohang, South Korea, in 2011.

From 2011 to 2019, he was a Senior Engineer with Samsung Electronics, Suwon, South Korea. In December 2019, he joined FuriosaAI, Inc., Seoul, where he is currently working as a Senior SoC Architect and a Design Engineer. His current research interests include large-scale SoC design and NoC interconnect for data center AI chips.

JUYUN LEE received the B.S. degree from the Department of Electronic Engineering, Inha University, Incheon, South Korea, in 2016, and the M.S. degree from the Graduate School of Convergence Science and Technology, Seoul National University, Seoul, South Korea, in 2021.

In 2018, he joined FuriosaAI, Inc., Seoul, where he is currently a Hardware Design Engineer. His research interests include digital systems and energy-efficient design for deep neural network processors.

YOUNGGEUN CHOI received the B.S. and Ph.D. degrees in electrical engineering from the Pohang University of Science and Technology, Pohang, South Korea, in 2010 and 2015, respectively.

From 2015 to 2018, he was a Senior Engineer with Samsung Electronics, Suwon, South Korea. In 2018, he joined FuriosaAI, Inc., Seoul, South Korea, where he is currently working as a Senior NPU Architect. His research interests include hardware acceleration for computer vision and large-scale language models.

MINHO KIM received the B.S. degree in electrical engineering and the M.S. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2000 and 2005, respectively.

He has held various design and verification engineer positions with STMicroelectronics, Geneva, Switzerland; Samsung Electronics, Suwon, South Korea; and SK Hynix, San Jose, CA, USA. He is currently a Principal Engineer with FuriosaAI USA, Inc., Santa Clara, CA, USA, and leading a team responsible for verifying accelerator designs.

CHANGJAE PARK received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 1994, 1996, and 2002, respectively.

He worked on design and verification of various high-speed serial interface IPs with Samsung Electronics, Hwaseong, South Korea, and SK Hynix, Seongnam, South Korea. Since 2020, he has been architecting high-speed interface for AI accelerators with FuriosaAI, Inc., Seoul, South Korea.

KISEOK JANG received the B.S. degree from the Department of Computer Engineering, Dong-A University, Busan, South Korea, in 2007.

In May 2021, he joined FuriosaAI, Inc., Seoul, South Korea, where he is currently a Hardware Design Engineer. His current research interests include PCI express and FPGA for data center AI chips.

YOUNGSIK KIM was born in Daegu, South Korea, in 1987. He received the B.S. degree in electrical engineering and computer science (double major) and the M.S. and Ph.D. degrees in electrical engineering from the Pohang University of Science and Technology, Pohang, South Korea, in 2009, 2011, and 2016, respectively.

From 2016 to 2019, he was a Senior Engineer with Samsung Electronics Company Ltd., Suwon, Gyeonggi, South Korea. In July 2019, he joined FuriosaAI, Inc., Seoul, South Korea, where he is currently working as a Design Verification Engineer.

YONGSEUNG KIM received the B.S. degree in information telecommunication engineering from Tongmyong University, Busan, South Korea, in 2007.

From 2007 to 2021, he developed the system software for AP and CP with Samsung Electronics, Hwaseong, South Korea. He is currently working on dynamic thermal management for AI accelerator SoCs with FuriosaAI, Inc., Seoul, South Korea.

CHANGMAN LEE received the B.Eng. and M.Eng. degrees in mechanical engineering from Korea Maritime University, Busan, South Korea, in 2005.

He is working on the Linux device drivers, such as PCIe and others on the ARMv8 and x86.

HYUCK HAN received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Seoul National University, Seoul, South Korea, in 2003, 2006, and 2011, respectively.

From 2011 to 2013, he worked with Samsung Electronics, Suwon, South Korea, as a Senior Engineer. In March 2014, he joined Dongduk Women's University, Seoul. His research interests are operating systems, database systems, and distributed systems.

WON EUNG KIM received the B.S. degree from Aviation College, Goyang, South Korea, in 1987, and the M.S. degree from Yonsei University, Seoul, South Korea, in 1990.

He worked as a Multiprocessor System Engineer with Samsung Electronics, Suwon, South Korea, from 1990 to 1992. From 1993 to 2013, he had been working for power converter design, mobile PMIC system design, and product manufacturing with Texas Instruments, Dallas, TX, USA. He joined FurosaAI, Inc., Seoul, in 2020 as a System Power Design Engineer.

RUI TANG (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Northeastern University, Boston, MA, USA, in August 2006.

He is working as a Vice President of Marketing and Strategy with MSQUARE Ltd., Shanghai, China. He worked as the Chief Strategy Officer and a General Manager of FuriosaAI USA, Inc., Santa Clara, CA, USA, and as an Engineer with Apple, Cupertino, CA, USA; Oracle/Sun Microsystems, Santa Clara; and Qimonda, Williston, VT, USA. His research interests include VLSI design.

JOON HO BAEK (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2005 and 2009, respectively.

He is the Founder and the Chief Executive Officer of FuriosaAI, Inc., Seoul, South Korea. Before starting up FuriosaAI, Inc. in 2017, he developed semiconductor products—both hardware and software—with AMD, Orlando, FL, USA, and Samsung Electronics, Suwon, South Korea. As the Founder and the CEO of FuriosaAI, Inc., he sets the product vision and orchestrates the engineering development of the firm.