DFU-E: A Dataflow Architecture for Edge DSP and AI Applications

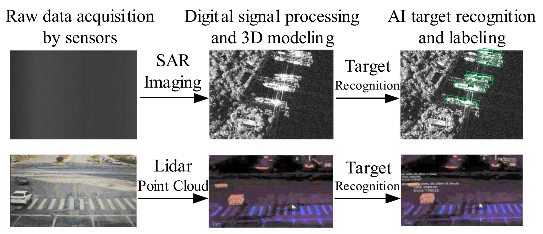
Wenming Li, Zhihua Fan, Tianyu Liu, Zhen Wang, Haibin Wu, Meng Wu, Kunming Zhang, Yanhuan Liu, Ninghui Sun, Xiaochun Ye, and Dongrui Fan

***Abstract*—Edge computing aims to enable swift, real-time data processing, analysis, and storage close to the data source. However, edge computing platforms are often constrained by limited processing power and efficiency. This paper presents DFU-E, a dataflow-based accelerator specifically designed to meet the demands of edge digital signal processing (DSP) and artificial intelligence (AI) applications. Our design addresses real-world requirements with three main innovations. First, to accommodate the diverse algorithms utilized at the edge, we propose a multi-layer dataflow mechanism capable of exploiting task-level, instruction block-level, instruction-level, and data-level parallelism. Second, we develop an edge dataflow architecture that includes a customized processing element (PE) array, memory, and on-chip network microarchitecture optimized for the multi-layer dataflow mechanism. Third, we design an edge dataflow software stack that enables automatic optimizations through operator fusion, dataflow graph mapping, and task scheduling. We utilize representative real-world DSP and AI applications for evaluation. Comparing with Nvidia's state-of-the-art edge computing processor, DFU-E achieves up to 1.42× geometric mean performance improvement and 1.27× energy efficiency improvement.**

***Index Terms*—Dataflow architecture, edge computing, digital signal processing, AI, multi-layer dataflow mechanism.**

I. Introduction

EDGE computing [1], [2] has emerged as a powerful tool for processing data and applications at the edge of a system, closer to where they are generated or used. This approach can reduce latency, improve performance, and enable real-time processing of data. Edge computing typically involves the deployment of specialized devices, including sensors, gateways, and servers, to facilitate the collection, processing, and analysis of data near its source. As depicted in Fig. 1, edge computing requires initial data collection followed by processing, as demonstrated in applications such as autonomous driving [3], [4], [5], [6], intelligent transportation [7], [8], [9], Synthetic Aperture Radar (SAR) imaging [10], and target recognition [11]. In the field of autonomous driving, source data is collected by lidar or cameras and then processed into images, which are subsequently sent to neural networks for target recognition and decision-making. In smart transportation, vehicle and people data are collected by millimeter-wave radar and cameras, and then sent to neural networks for recognition. In SAR imaging and target recognition, the target is scanned and imaged by radar, and the data is sent to the neural network for target recognition. All these applications share a common feature: data needs to be processed by a digital signal processing (DSP) module, then further processed by an artificial intelligence (AI) module, and finally, the processed results are output. This process is akin to obtaining external data through eyes and ears and transmitting it to the brain for identification and decision-making.


Raw data acquisition by sensors
Digital signal processing and 3D modeling
AI target recognition and labeling
SAR Imaging
Target Recognition
Lidar Point Cloud
Target Recognition
Fig. 1. Representative examples of edge applications.

The convergence of applications in the DSP and AI domains has given rise to new challenges in processor design [12], [13], [14], [15]. To address this convergence scenario, several approaches have been proposed, each with its unique strengths. One approach involves using a combination of different chips, with dedicated DSP chips handling DSP tasks and specialized NPU chips for AI processing. While these solutions can provide domain-specific optimization, the fixed ASICs fails to adapt to newly developed algorithms. For example, the characteristics (computing, memory access, parallelism, etc.) of new algorithms vary, which calls for hardware flexibility. Another strategy employs heterogeneous logic designs on a single chip, integrating both DSP processing cores and AI processing cores. This approach allows for tighter integration of DSP and AI functions, but it may face challenges in programming and communication. Facing cross-domain application scenarios, such as smart transportation, the heterogeneous solutions lack effective solutions to achieve low energy consumption and improve performance

Received 2 January 2024; revised 24 February 2025; accepted 24 March 2025. Date of publication 28 March 2025; date of current version 18 April 2025. This work was supported in part by National Key R&D Program of China under Grant 2023YFB4503500, in part by Beijing Nova Program under Grant 20220484054 and Grant 20230484420, in part by Beijing Natural Science Foundation under Grant L234078, and in part by the CAS Project for Youth Innovation Promotion Association. Recommended for acceptance by R. Tolosana. *(Corresponding author: Zhihua Fan.)*

The authors are with the State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing 100049, China, and also with the School of Computer Science and Technology, UCAS, Beijing 100190, China (e-mail: liwenming@ict.ac.cn; fanzhihua@ict.ac.cn; liutianyu@ict.ac.cn; wangzhen@ict.ac.cn; wuhaibin@ict.ac.cn; wumeng@ict.ac.cn; zhangkunming@ict.ac.cn; liuyanhuan@ict.ac.cn; yexiaochun@ict.ac.cn; fandr@ict.ac.cn).

Digital Object Identifier 10.1109/TPDS.2025.3555329

on the complex hardware. The third option is the multi-domain acceleration solutions, such as NVIDIA GPUs [16], [17], [18], [19], which have garnered increasing attention and adoption. These accelerators offer a versatile and programmable hardware architecture that allows for the execution of both DSP and AI tasks with the Single Instruction Multiple Thread (SIMT) execution model. This design choice leverages the inherent parallelism in both DSP and AI algorithms, enabling more efficient utilization of computational resources and adaptable processing capabilities. Considering factors such as system stability, area constraints, power consumption, and the dynamic nature of algorithms, general-purpose acceleration solutions demonstrate significant advantages [20]. Notably, their flexible architecture enables the allocation of hardware resources according to the specific computing power requirements in different DSP and AI scenarios, thereby achieving higher utilization and performance.

In this paper, driven by real-world requirements from edge DSP and edge AI applications, we introduce a dataflow-based accelerator, DFU-E, designed for edge computing. The key contributions in this paper are:

- We analyze the characteristics of DSP and AI applications and summarize crucial indicators for edge computing, which helps us in architecture design.
- We introduce DFU-E, a general-purpose accelerator designed for edge computing based on novel dataflow execution model, especially for DSP and AI applications.
- We propose a multi-layer concurrent dataflow model to enhance execution parallelism and improve hardware utilization. This model allows efficient execution of multiple dataflow graphs (DFGs) simultaneously, which boosts performance and reduces power consumption.
- We implement various hardware and software optimizations, including powerful single instruction multiple data (SIMD)/Tensor engines, forwarding network-on-chip (NoC), hybrid scratchpad memory (SPM) and Cache memory, fine-grained power management and optimized compiler tool chain, resulting in improved performance and energy efficiency.
- Evaluated with typical edge computing workloads, DFU-E outperforms the NVIDIA edge computing processor, with higher improvements in performance and energy efficiency.

This paper is organized as follows. Section II discusses the challenges brought by edge applications. Section III presents the DFU-E architecture. Section IV discusses our experimental methodology and results. Section V shows the related works. We finally conclude the paper in Section VI.

II. Challenges Brought by Edge Applications

With the coming end of Moore's Law [21] and Dennard Scaling [22], it's hard to continuously improve the computing performance of general-purpose processors. Application-specific integrated circuits offer great advantages because they can process one or several applications more efficiently with a smaller area and less power [23]. However, applications evolve faster than processors, making existing hardware face difficulties

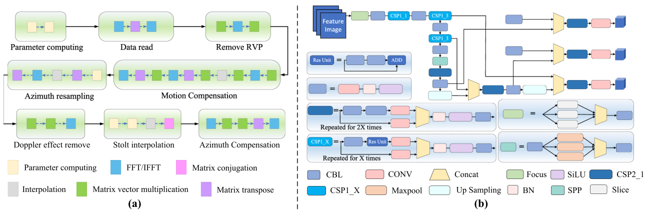


Fig. 2. Execution flow of representative DSP and AI applicaitons, SAR imaging(a) and YOLO neural network(b).

when adapting to newly developed algorithms. Especially for application scenarios like edge computing, there are high requirements for computing power, flexibility, and power consumption.

Fig. 2 shows typical algorithm flows of SAR imaging and YOLO neural network [24]. First, as a widely applied algorithm in a radar system, SAR imaging contains a series of different data processing steps [25], including FFT/IFFT, vector-matrix multiplication, interpolation, matrix transpose, and matrix conjugation. In this case, specified accelerators obviously are not compatible to every step of such complex algorithm types, let alone the massive data throughput, real-time and energy efficiency requirements. Another typical example is convolutional neural network (CNN) [26], [27], [28], [29], the input or batch size of each layer may be different, which leads to different demands of computing structure. The characteristics of these applications can be concluded as follows: 1) Large-scale regular data with simple calculation patterns, such as matrix multiplication, convolution, and FFT. The input data in these applications are regular, which means the source data has good temporal or spatial locality, and the calculation pattern is regular. Therefore, the designers can leverage high-width SIMD structures to speed up such kinds of applications. 2) Small-scale regular data with a simple calculation pattern, which is not large enough to fully utilize the SIMD hardware resource. What's worse, the energy efficiency of such applications will be poor due to low utilization. 3) Large-scale regular data with complex calculation patterns, which may mix with conditional jump, swap, comparison or other operations, is more complicated than simple operations, making it challengeable to fully exploit the power of SIMD. 4) Small-scale regular data with complex calculation patterns, which likes small-scale regular data with simple calculation patterns. For these applications, the most effective way is to exploit the parallelism of data execution as much as possible with multi/many-core processors. It is not enough to use heterogeneous ASIC computing systems to solve the above problems, because it will bring programming problems and introduce excessive system hardware complexity, resulting in non-negligible communication overhead [30].

In addition to being able to efficiently support multiple types of computing modes mentioned above, it is necessary to maintain low power consumption while increasing hardware flexibility to adapt to changing algorithms. Dataflow architectures [31], [32], [33], [34], [35], [36] are promising approaches because of their ability to exploit multi-level parallelism while achieving high hardware utilization and energy efficiency, which is very

TABLE I
HARDWARE INNOVATION AND SOFTWARE ADAPTION MADE FOR DFU-E TO CONFRONT THE DEMANDS OF EDGE WORKLOADS

Characteristics of Applications	**Hardware Innovation**	**Software Stack**
High energy efficiency and strong real-time requirements in edge computing scenarios	Dataflow execution model to reduce control complexity and memory access requirements	Dataflow runtime system to improve real-time scheduling of DFG
Low latency requirements for data transmission and task execution	NoC supports forwarding transmission to reduce data transfer delay	Support user- controlled memory distribution
Frequent synchronization requirements for parallel execution of complex data and instructions	Using dataflow to avoid synchronization operations, which is solved when generating DFG	Dataflow compiler to generate DFG and map to PE array
Matrix and convolution computations operations are the main types of operations	Support SIMD unit and Tensor unit with different operation precision	Compiler generates SIMD or Tensor codes, supporting mixed precision fusion execution
The data size of matrix and convolution operations is not fixed	Hardware reconfigurable to support different width SIMD execution models	Supports automatic selection of SIMD execution model
Transcendental function for DSP and activation functions in CNN	Special function unit (SFU) for processing transcendental functions	Dedicated SFU instructions
Data with high sparsity is often observed in CNN's inputs and intermediate values	Support sparse computing with Tensor unit, enable sparse data decompression during transfer with DMA	Libraries of sparse tensor computing
Batch processing with concurrent tasks often shares same data	Enable data broadcasting to sPEs and flexible packet combination and splitting in NoC	DFG defined data sharing and transmission
Irregular data access in some applications	Hybrid reconfigurable SPM/Cache on-chip memory	Support User-configured SPM or Cache
Kernel code loading affects overall DSP and CNN performance	Enable kernel prefetch to control buffer	Support user-controlled kernel code prefetch
Different workloads result in various computing and power requirements	Efficient power managements based on on-demand adjustments	Power management policy algorithm

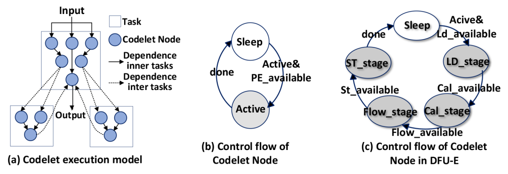


Fig. 3. Codelet dataflow model vs DFU-E dataflow model.

suitable for edge computing scenarios. Compared to the SIMT execution model in NVIDIA GPUs, dataflow architectures can offer higher parallel execution ability and reconfigurability, since dataflow architecture is driven by data, not instructions, they can better accommodate the dataflow characteristics and data dependency relationships of different applications. This makes dataflow architectures perform more efficiently in handling complex data dependencies and irregular computation processes. Additionally, dataflow architectures can optimize hardware resources utilization by mitigating resource wastage associated with thread synchronization and data conflicts, resulting in enhanced energy efficiency. Numerous studies have analyzed the characteristics of dataflow and SIMT structures [37], [38], [39].

In DFU-E design, we implement an optimized Codelet model [40], as shown in Fig. 3. In the Codelet model, the execution of the node adopts a non-preemptive mechanism. When a node is executed, the node will occupy the PE resources, and the PE will not be released until all the instructions in the node are completed. DFU-E decouples the task within the dataflow graphs (DFG) node into four consecutive pipeline stages. Each stage is an atomic schedule and execution unit. In this way, a PE can be shared by at most four different DFG nodes at the same time and the memory access and data transfer latency can be overlapped as much as possible. Table I summarizes the design of DFU-E's hardware and software based on the requirements of the applications, including execution model, PE Architecture, network, memory, etc.

III. DFU-E ARCHITECTURE

Based on the design considerations discussed above, we innovate DFU-E with novel hardware/software enhancements, as outlined in Table I. More details are as follows.

A. *Overview of DFU-E*

Fig. 4(a) is the overall architecture of DFU-E, which consists of a PE array with 16 PEs, a global on-chip memory divided into 16 banks, two data recombination modules, two DMA channels, a PCIe 4.0 interface, eight 32 b-wide LPDDR5 interfaces, an MCU and a Micro Controller (MiCC) module. The PE array is organized as a 2D mesh by 16 routers and can be reconfigured as an 8×8 sPE array, which is described in detail next section. The DFG can be mapped, scheduled, and executed with these PEs or sPEs. Two DMAs are responsible to transfer data from DRAM to on-chip memory. The MCU is a 4-core RISC-V CPU used to manage the execution of DFU-E. The MiCC is responsible for run-time schedule on the whole PE array scale, including kernel switch, task-level parallelism exploiting, as well as instantiation of block execution. In order to improve energy efficiency, we have implemented a distributed power management mechanism, including the Local PCU (in dark blue) and Local Alarmer (in yellow) units in Fig. 4(a). The detailed introduction of DFU-E is as following sections.

B. *Architecture of sPE*

As described above, each PE in DFU-E consists of a cluster of four sub-PEs (sPE): sPE(1) to sPE(4), shown in Fig. 4(f). The four sPEs are not simply interconnected through a common bus. It is a tightly coupled inter-pipeline interconnection, which

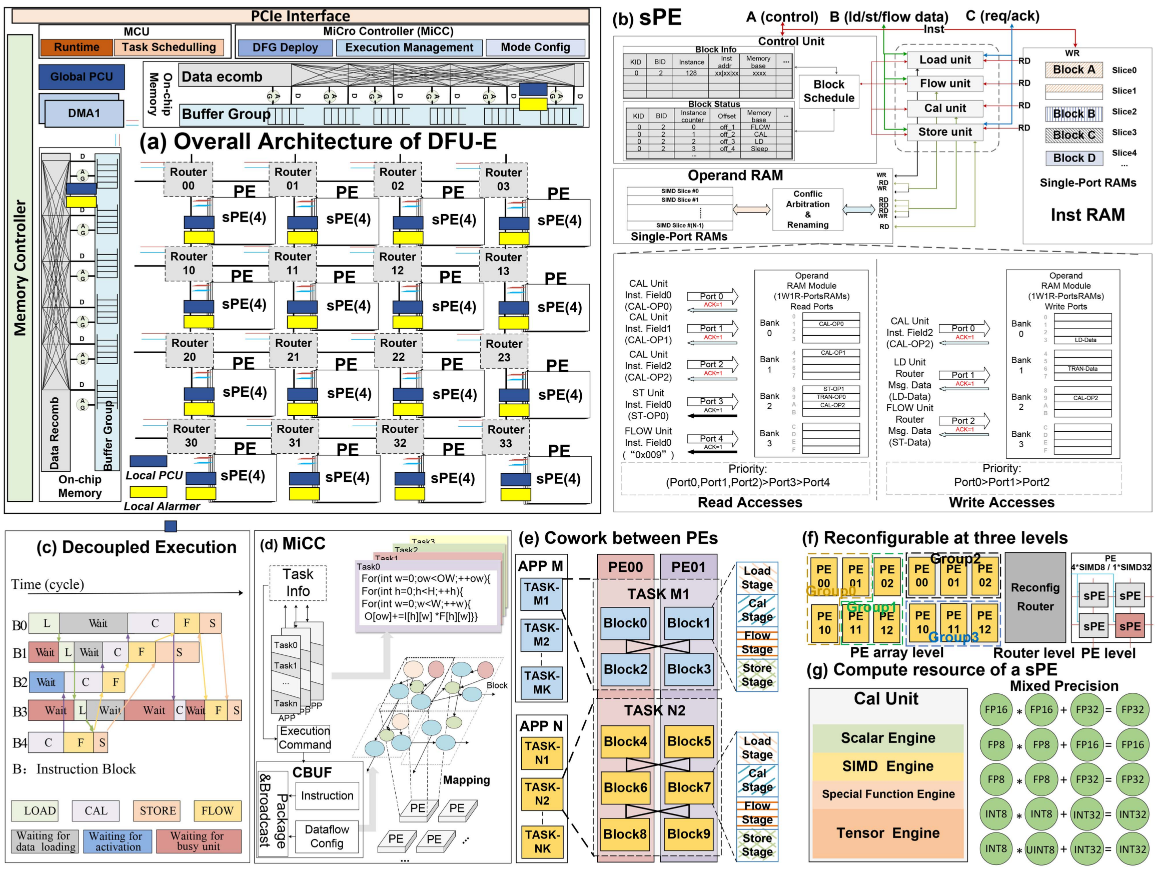


Fig. 4. Overview of our DFU-E and the internal structure of a sPE.

means the computing units of four sPEs can be merged as a high-width computing unit. For example, each sPE has a SIMD-8 computing unit, therefore four sPEs can be merged as a SIMD-32 computing unit. The PE array can be configured to work in two modes, a small-scale 4× 4 PE array or a large 8× 8 PE array. Therefore, according to the requirements of workloads, PE or sPE can provide high-width SIMD or Tensor computing resources or low-width SIMD or Tensor computing resources to pursuit higher energy efficiency. Fig. 4(b) is the detail description of a sPE, including:

- A Control Unit, which is responsible for configuring sPE, maintaining the execution status, and scheduling blocks that are mapped to the sPE. A series of combinational logic forms block matching mechanism, prefetching blocks for the decoupled pipeline before execution. Furthermore, an active and ack control protocol with block tokens and credits enables coarse-grained pipelined execution of blocks inner sPE and keeps dataflow execution dependencies inter sPEs. The reconfigurable hierarchical control mechanism makes it possible for DFU-E to deal with many kinds of computation manner and task-level parallelism. Unlike other processors that use a centralized control scheme, DFU-E uses a distributed control scheme in which each sPE has its own control unit and can make most of the control decisions independently. With this capability, it can enable fine-grained control and data reuse, which are important to achieving the desired programmability.
- Four decoupled function units, named Load Unit, Cal Unit, Flow Unit, and Store Unit. Instructions are arranged and decoupled into four sequential processing stages: Load, Cal, Flow, and Store. Therefore, data feeding is separated from computing. Data are loaded prior to execution, which reduces the underutilization due to lack of data and parallelism improvement of PE components. The Load Unit loads data from on-chip memory to Operand RAM. Once the associated task is enabled, an instruction block can execute its load operation. The Cal Unit represents the execution unit. An instruction block is ready to fire to the execution unit after its dependent data are all satisfied and activate signal arrives. Under the scheduling of Control Unit, these blocs with status information, like memory address, head PC, and instance counter, are dispatched to four corresponding function units once they come with an idle state. The Store unit is responsible to store calculation results to on-chip memory. The Flow unit transfers data between PEs or sPEs. As is shown in Fig. 4(c), such a multi-step/stage management method overlaps the steps and stages of different instruction blocks. This in turn helps maximize the utilization of the PE execution units, especially Cal Units, which is important because improving the utilization of Cal Units is the key to improving performance and energy efficiency.
- A data buffer and an instruction buffer, named Operand RAM and Inst RAM. Under the same capacity, SRAM has a smaller area and lower power consumption than the register. So

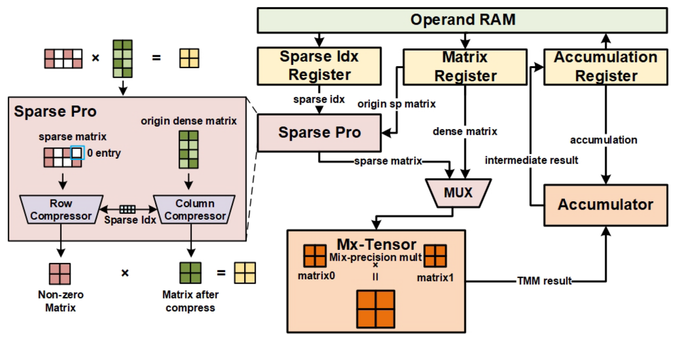


Fig. 5. The execution flow of tensor computation.

we replace the large register file with SRAM and use hardware and software mechanisms to guarantee that this SRAM-based approach can achieve performance similar to that of the RF-based approach. The Inst RAM consists of multiple single-port SRAM banks. Each bank can only be occupied by a single function unit at any time. The Operand module consists of multiple 1-write-1-read SRAM banks, each of which can serve at most one write and one read at the same time. These four function units fetch operands from a shared operand ram which can handle five read accesses and three write accesses following a priority: Cal >Load, Store >Flow. As shown in the bottom of Fig. 4(b), three out of five read ports (Ports 0–2) are used by the CAL Unit. These three ports have the highest priority and must be served at the same time. The ports of Store and Flow Units have a lower priority. As a result, CAL Instructions can always be served without stalling, but ST and Flow Instructions may have to wait because of bank conflicts. To avoid the conflict of CAL Unit read ports, we map input operands of a CAL instruction to be distributed in different banks during compilation.

C. *Cal Unit*

In order to meet a wide variety of computing requirements, the Cal Unit supports scalar, SIMD, tensor, and special function computations, while supporting FP64, FP32, INT32, FP16, FP8, INT8, and UINT8 computation precisions. For SIMD unit, it can be reconfigured to support different precisions, and a flexible shuffling unit is equipped for SIMD reordering and merging, to satisfy different requirements of vector processing patterns. In AI workloads such as YOLO and Transformer, a large number of GEMM operations take up the majority of the workload's execution time. The tensor ALU is more suitable for handling such computations. The tensor ALU is configured with a matrix register, accumulation register, sparse index register, and sparse processing module to accelerate dense or sparse GEMM operation.

Fig. 5 shows the detailed procedure of GEMM operations performed by the tensor unit. Before the computation, the multipliers and addends of the multiplication-add operation are loaded from the Operand RAM into the matrix register. For the sparse GEMM, the index of the sparse matrix needs to be loaded into the sparse index register. A sparse processing module is used to compress the original matrix, and the original data is entered into the Mx-Tensor (Mixed-precision Tensor) Unit after a MUX. The MX-Tensor Unit supports mixed-precision floating-point/fixed-point GEMM operations. After the MX-Tensor Unit completes the calculation, the results are temporarily stored in the accumulation register for data reuse. By repeating the above steps, the tensor unit can perform larger GEMM operations, and the final result is stored back in the Operand RAM by the accumulation register.

The advantage of our design is that the matrix sparsity is allowed to be processed inside the tensor unit. In the load stage, the sparse index table is generated according to the position of zero entries in the sparse matrix and then loaded into the sparse index register by the Operand RAM. As shown in Fig. 5, at the beginning of computation, the sparse processing module performs row compression/column compression on the two source operand matrices respectively according to the sparse index and finally forms a matrix that meets the shape of the Mx-Tensor unit. Provided that the data processed every time does not exceed the capacity of the matrix register, our sparse processing module can be configured to compress the matrix in different ratios, such as 4:2 or 4:1, which makes the sparse processing able to satisfy the needs of more scenarios. This design allows tensor units to support multiple types of sparse and dense GEMM without introducing additional storage resources, and sparse GEMM can achieve up to 4 times better performance than dense GEMM.

D. *Micro Controller (MiCC)*

The control flow and structure of MiCC are shown in Fig. 4(d). MiCC is responsible for task scheduling and assignment. After the compiler divides the applications into several tasks, each task forms a DFG, containing instruction blocks with data dependencies that are to be mapped to the PE array for execution. All the DFG configurations and binary files are transferred to Control Buffer (CBUF) through DMA before being packaged and broadcasted to PEs or sPEs. During the execution, MiCC is responsible for the run-time schedule on the whole PE array, including kernel switch, task-level parallelism exploiting, as well as loop execution of blocks.

E. *Flexible NoC Design*

Fig. 4(b) describes the structure of a sPE. There are three separate I/O ports, a control port (A), a memory access port (B), and a *REQ* and *ACK* port (C). The three ports are connected to the three sets of separate networks of the NoC. The A network is responsible for transporting instructions and configuration commands to each sPE, therefore, it is unidirectional between configuration unit and PEs. The data transmission of *LOAD*, *STORE* and *FLOW* operations go through the memory access lane, which is the B network. And the C network is used for the request commands of memory access and the *ACK* information to maintain the execution sequence between DFG nodes, so they are bidirectional.

As shown in Fig. 6, DFU-E configured as an 8×8 sPE array, the NoC of DFU-E is designed with forwarding connections to improve data fetching bandwidth and alleviate transfer latency. Routing ports of each sPE are connected with the adjacent sPEs,

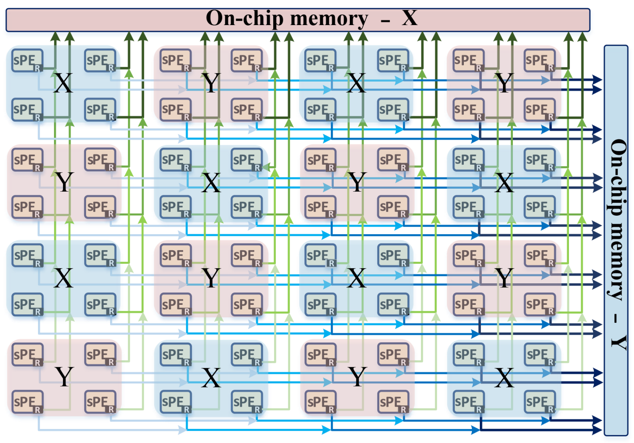


Fig. 6. Forwarding networks.

while the connections are also forwarded straight ahead. Physically, each data path in a certain direction passes through the adjacent sPE to the next unit (sPE or memory), linking two inputs of them. As the enhancement of the mesh Noc, this interlaced forwarding interconnection allows two concurrent data transfers in rows and columns. Application kernels described as DFGs with different data reusability can benefit from the forwarding NoC. As the 8 × 8 NoC shown in Fig. 6, memory-bound workloads suffering from long distance will benefit from lower transfer latency because all memory requests and returns are transferred in a cluster way (4 × 4 NoC), which results in the half reduction of transfer hops in NoC. For example, for the network in the *X* and *Y* directions, the closer to memory, the greater the degree of transmission congestion. To alleviate the congestion, we can map the nodes of DFG with low data correlation to the clusters of *X* and *Y* respectively, and fetch data from the memory of *X* and *Y* respectively. Transfers of data dependence between nodes in the complicated DFG with high data reusability, are also accelerated since multi-hop data dependence is flowed along the tighter forwarding interconnection.

F. Enhanced DMA Design

To solve the problem of excessive consumption of CPU resources caused by the transfer of large amounts of data, we have introduced an enhanced DMA module for direct data transmission. Based on this, DMA can dynamically perform data preprocessing tasks according to configuration requirements, such as matrix regularization, transposition, and padding, to reduce the operating burden of computation units and fully exert its computing power. Unlike ordinary DMA which only performs simple transfers, our enhanced DMA can reduce the design complexity of the computation unit through the real-time preprocessing function mentioned above, allowing data to be directly processed when it reaches the computation unit, thereby improving computing efficiency.

To improve the utilization rate of hardware resources, the execution order of the data transfer and data computation can be reasonably scheduled to achieve the effect that while the computation unit is processing, the data transfer component is

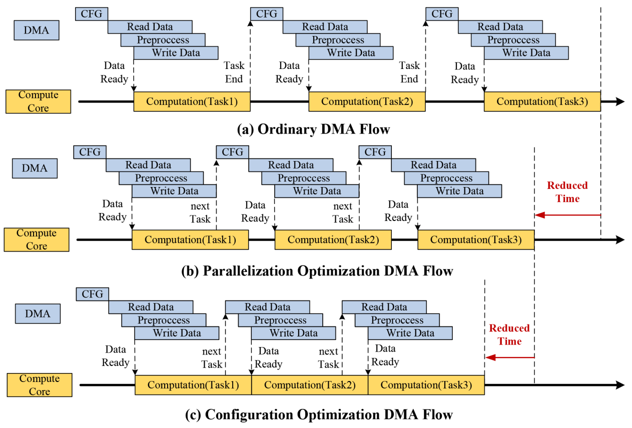


Fig. 7. Process flow of the enhanced DMA.

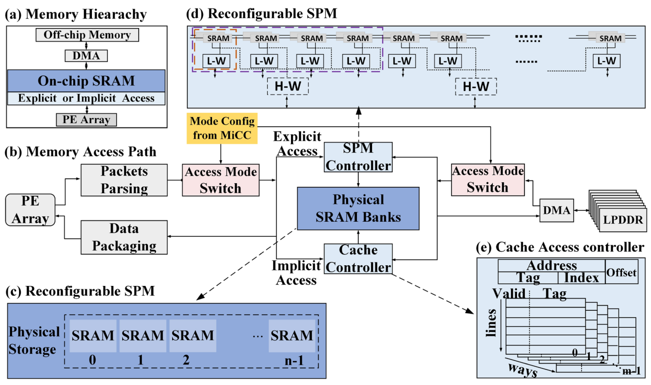


Fig. 8. Hybrid storage of SPM and cache.

transferring the next batch of data, thus minimizing time delay. Fig. 7 shows the effect of parallelization optimization of our enhanced DMA during data transmission and computation.

In order to reduce the time overhead of DMA configuration, our enhanced DMA can pre-store the configuration information of repeated DMA transfers, and then drive DMA to read and parse the configuration information autonomously. The bottom of Fig. 7 shows the effect of optimizing the pre-stored configuration of DMA. On this basis, for regular data transmission, DMA can open the address accumulation function required for a new round of transmission by itself without repeatedly storing configuration information in on-chip storage, further saving storage space and register configuration time and thus achieving higher performance.

G. Hybrid On-Chip Memory

In Fig. 8(a), the on-chip memory of DFU-E applies a hybrid architecture combining SPM and cache to support both explicit and implicit data movement while sharing the same physical SRAM on-chip, which brings low area overhead. As shown in Fig. 8(b), the running mode of on-chip memory is configured by MiCC when starting a new round of execution. SPM data path uses a dedicated multi-functional DMA that can efficiently transfer a large amount of data from LPDDR in a *Ping-Pong* way to overlap latency between computation and data fetching. This kind of transfer is static, explicit and suitable for applications involving regular memory accesses, which can be controlled by developers.

Cache usually consumes more energy and time because of tag search and comparison, as shown in Fig. 8(e), but it possesses the function of dynamic and off-chip global memory access, which is transparent to developers and suitable for applications involving more irregular memory access. Because the on-chip Cache is shared with the whole PE array, there is no need to maintain consistency between Cache blocks. We adopted the write-through Cache pattern, therefore, when the current execution is finished, we just invalidate all the Cache lines if the next round of execution uses fresh data transmitted by PCIe. This ensures that the new round of execution will not be effect by previous data residing in Cache. In DFU-E, Cache is organized in a set-associative way. Different from traditional Cache, our Cache uses the interleave mapping method when mapping DDR space, which is convenient for PE to access the corresponding Cache through NoC routing.

H. Power Management

Tens of billions of transistors have been integrated into the DFU-E chip. High concurrent circuit flipping will cause serious instantaneous current, which leads to severe IR drop problems. Excessive IR drop may cause timing failure, abnormal reset, or disruption of data processing. Therefore, the power design of DFU-E revolves around power stability and energy efficiency. As shown in Fig. 4(a), the DFU-E power control system consists of a Global Power Control Unit (Global PCU), Local Power Control Unit (Local PCU), and Local Alarmer, achieving a two-level power optimization realtime management structure. Global PCU allocates real-time IR drops and temperature thresholds of sub-modules based on the application type and current performance bottlenecks of each sub-module, maximizing and reasonably allocating applications from a system perspective. The Local PCU reads the the operating frequency, IR drop threshold, and temperature threshold configuration from the Global PCU and the corresponding IR drop, temperature, and performance information of the local alarm in real-time, efficiently configuring the corresponding sub-modules in real-time, without communicating with the Global PCU, ensuring power stability and high energy efficiency in real-time. Local alarmer can quickly sense IR drop and temperature information from the Local PCU, ensuring efficient subsystem regulation.

As shown in Fig. 9, in a new task configuration phase, the Global PCU analyzes the current system bottlenecks based on the execution information of the DMA, on-chip memory, and PEs. Then, based on the characteristics of the upcoming DFGs and the ratio of busyness duty cycles, IR drop alarm and temperature alarm, it issues IR drop threshold, temperature threshold and frequency configuration to Local PCU configurations to achieve better choices at the system level. When the Local PCU generates an alarm, the Global PCU adjusts the system threshold configuration to improve chip stability. Local PCU receives Global PCU configuration information and changes operating frequency considering the IR drop, temperature, and performance information of the Local Alarmer, to maintain

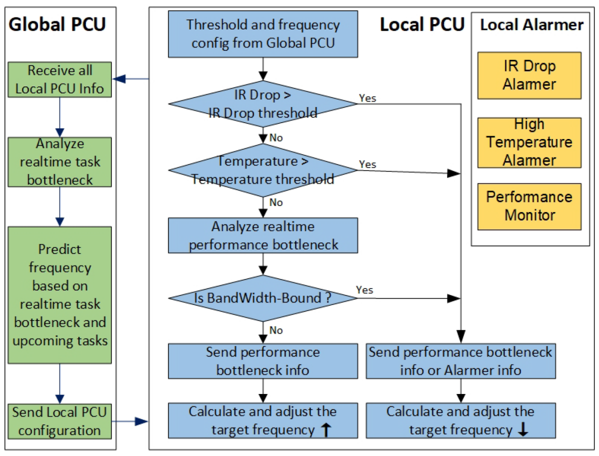

Global PCU
Receive all Local PCU Info
Analyze realtime task bottleneck
Predict frequency based on realtime task bottleneck and upcoming tasks
Send Local PCU configuration
Threshold and frequency config from Global PCU
Local PCU
IR Drop > IR Drop threshold
Yes
No
Temperature > Temperature threshold
Yes
No
Analyze realtime performance bottleneck
Is BandWidth-Bound ?
Yes
No
Send performance bottleneck info
Send performance bottleneck info or Alarmer info
Calculate and adjust the target frequency ↑
Calculate and adjust the target frequency ↓
Local Alarmer
IR Drop Alarmer
High Temperature Alarmer
Performance Monitor

Fig. 9. Power control strategy of DFU-E.

the system works in a dynamic, efficient, and stable status in real-time. The Local Alarmer can provide real-time feedback on IR drop risks, high-temperature risks, and ensure realtime performance. When the local IR drop or temperature exceeds the threshold, the Local PCU will emit warnings to the Global PCU to achieve the secondary dynamic feedback configuration.

I. Multi-Level Parallsim

In order to deeply exploit the parallelism of data processing for edge computing applications, we design the reconfigurable dataflow architecture by fully supporting multi-level parallel processing, including TLP (Task-level Parallisim), BLP(Block-level Parallisim), ILP (Instruction-level Parallisim) and DLP (Data-level Parallisim). A dataflow architecture with configurable width of SIMD and tensor structure is proposed to support DLP. A decoupled pipeline is implemented to improve BLP and ILP. And multiple DFG tasks can be mapped and scheduled to the same PE groups or different PE groups to exploit TLP.

- **TLP**, as is shown in Fig. 4(e), multiple applications can run on DFU-E concurrently. Each application consists of a sequence of consecutive tasks. Tasks from two different applications are mapped to *PE00* and *PE01*, which means that DFU-E supports parallel execution at task level. Each task is an independent DFG. Since more tasks executed in a PE in parallel means more hardware resource requirements, DFU-E hardware supports up to four tasks mapped to the same PE simultaneously.
- **BLP**, as described above of TLP, each task consists of multiple execution blocks (instruction block) that are organized in a dataflow manner. After one or more task are mapped to a PE, more instruction blocks are scheduled and executed in the same hardware resource, which can hide data waiting delays from each other. Since the execution components are decoupled into LD, CAL, FLOW, and ST, all blocks are executed in a competitive manner. The instructions of each block consist of up to four consecutive execution stages, which are LD Stage, CAL Stage, FLOW Stage, and ST Stage.
- **ILP**, the decoupled four function units in DFU-E equal to a four-issue pipeline processor. different stages of instructions are decoded and issued separately. Fig. 4(b) and (c) show the

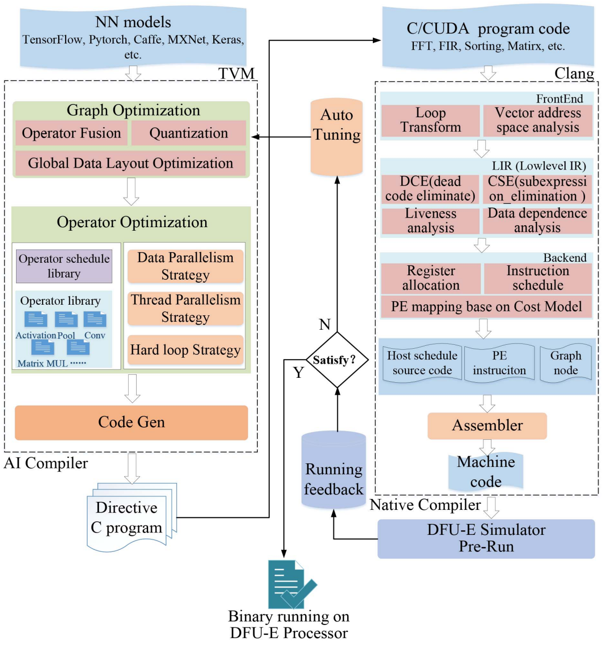


Fig. 10. Compilation flow of DFU-E compiler.

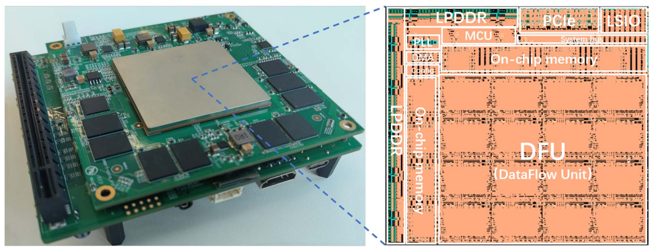


Fig. 11. DFU-E board system and chip floorplan.

hardware structure and execution example of instruction-level parallelism.

- **DLP**, for applications with simple calculations and large data blocks, SIMD is an efficient data-level parallel execution mode, and for matrix and convolution operations, the tensor is an efficient data-level parallel execution mode. In DFU-E, we not only implement high-width SIMD and tensor execution hardware but also support SIMD and tensor reconfigurable in width to adapt to more application scenarios, which is described in Section III-C and Fig. 4(g).

By efficiently utilizing TLP, BLP, ILP, and DLP, DFU-E can make full use of hardware resources and cope with different application workloads, resulting in enhanced performance and reduced resource overhead. One key advantage of DFU-E is its inherent lack of the need for explicit synchronization mechanisms. In the SIMT execution model of NVIDIA GPU, threads must synchronize frequently, introducing delays and resource contention. However, in DFU-E's dataflow execution model, data synchronization is resolved when the DFG is generated, eliminating the need for explicit synchronization.

J. *Compilation and Mapping*

DFU-E is based on a Codelet-like execution model, a coarse-grained (instruction-block level) dataflow execution model where inner-block execution is split by the decoupled pipeline. Based on the DFU-E's dataflow execution model and multi-level parallel execution features, we implement an optimized compiler to generate efficient DFG code for DFU-E, which is comprised of AI compiler, native compiler, and highly optimized kernel libraries, as shown in Fig. 10. The AI compiler is implemented based on the TVM framework. For the neural-network applications constructed by PyTorch, TensorFlow, MxNet, etc., the frontend of the AI compiler will transform it to computational graph IR (intermediate representation). Then automatic operator fusion, quantization and data layout optimization are adopted to generate more optimized graph IR. For graph nodes of a single operator, such as a convolution or a relu kernel, manual tuning based on experience or automatic tuning will assist developers to generate optimized directive `C` program. The automatic tuning will surf through the search space consisting of schedule-specific parameters such as loop unrolling factors, loop split factors, data parallelism dimension, to get the an optimized schedule strategy. Of course, we provide a series of deeply optimized AI operator libraries for developers to call, similar to NVIDIA's cuDNN.

The native compiler provides two sets of programming interfaces to the operator developers, user-friendly `C`-style language and CUDA language. Developers can write a kernel function by simply adding some `#pragma` directive which represents hardware loop, thread parallelism, and data parallelism to `C`-style language kernel function. The frontend will transform `C`-style operator and CUDA operator to unified LIR, Low-Level Intermediate Representation, with hardware-related features.

At the native compiler middle end, machine-independent optimization such as dead code elimination, liveness analysis, common subexpression elimination and data dependence analysis will reduce the computational time. At the native compiler backend, machine-dependent optimization such as register allocation, and instruction schedule will generate more efficient target code. At the graph node mapping phase, we implement a cost model that can predict the performance when the target code running on our DFU hardware. Finally, we support a simulator for users to run and debug applications. When the simulation finish, some profiling information such as running time, and resource utilization will feed back to the AI compiler to automatic adjustment the optimization strategy.

IV. Evaluation

To demonstrate the effectiveness of DFU-E's hardware and software, we validated its performance, efficiency, and flexibility, respectively. The chip and floorplan of DFU-E are shown in Fig. 11.

A. *Experimental Setup*

We developed a dataflow simulator based on the SimICT parallel framework [41]. This simulator is primarily used to verify correctness and assess performance and computational component utilization. It simulates behaviors such as computation, memory access, and instruction conflicts. Additionally,

TABLE II
SPECIFICATIONS OF THE BENCHMARKS

Benchmark	Kernel	Precision	Data Scale
FFT	FFT/iFFT	fp32/fp64	length: 256 to 1M
GEMM	General Matrix Multiplication	fp16/fp32	scale: 4 to 2048
Convolution	CONV	fp32/fp16	kernel:1×1,3×3,5×5
SAR	Para&data read	int32	2k×2k 4k×4k 8k×8k
	Remove RVP Motion Compensation Doppler&RefFun Stolt Interplation Azi Compensation	fp32/fp64	
YOLO-v5s	Backbone (Conv 1, 2, 4) Neck (Conv 6) Prediction (Conv 8)	fp16/fp32	3×640×640
CRNN	Convolutional Layers Recurrent Layers	fp16/fp32	3×32×320
Transformer -VIT	Patch_embedding Multi_Head Attention MLP Encoder	fp16/fp32	batch_size: 64/512 kernel: 256×256
Laser Radar	Scan&Filter 3D reconstruct Spatial coordinate	fp32	DBNet

TABLE III
TECHNICAL SPECIFICATIONS OF ACCELERATORS FOR EVALUATION

	Jetson Orin	DFU-E
Architecture	Ampere	Dataflow
FP64 (TFLOPS)	2.7	2.4
FP32 (TFLOPS)	5.3	4.8
FP16 (TFLOPS)	**85 (Sparse/Tensor)** 10.6 (CUDA)	**76.8 (Sparse/Tensor)** 9.6 (Vector)
FP8 (TFLOPS)	-	150 (Sparse/Tensor)
INT8 (TOPS)	**170 (Sparse/Tensor)** 105 (Sparse/DLA)	**153.6 (Sparse/Tensor)** 19.2 (vector)
On-chip Memory (MB)	11 (GPU Cache)	8 (Hybrid SPM/Cache)
Frequency (GHz)	1.3	1.2
Memory (GB)	64	64
Bandwidth (GB/s)	204.8	204.8
Board TDP (Watt)	15-60	12-65
Area (mm^2)	-	416
Chip Technology (nm)	8nm	12nm
Interconnect	PCIe4.0	PCIe4.0

we implemented the modules of the dataflow architecture in Verilog using the Synopsys tool. We synthesized the design using Synopsys Design Compiler and an industrial 12 nm GP standard VT library, meeting timing at 1.2 GHz. To ensure accuracy, we calibrated the simulator's latency error to within 7% by comparing the Verilog implementation with the functional correctness of the C simulator. First, we verify the computational results and functional correctness of the simulator in C and the implementation in Verilog. The error here refers to the total latency error of the test program in the C simulator and Verilog environment. Since the latency of task switching, as well as the latency of pipeline blocking, is very difficult to ensure consistency. We also used Synopsys PrimeTime PX for accurate power analysis based on netlists and application waveforms. Table III shows the hardware parameters.

The benchmarks we evaluated are all abstracted from DSP and AI applications adopted in various scenarios, as listed in Table II. Above all, two critical kernels, FFT and CONV with various data scales in DSP and AI processing are examined to illustrate the flexibility and utilization of DFU-E hardware. We make experiments on the Tensor Unit, network-on-chip, and memory, to show the efficiency of our design. In order to cover the thorough imaging-and-detecting application scenario in edge processing, we first select the SAR algorithm whose processing

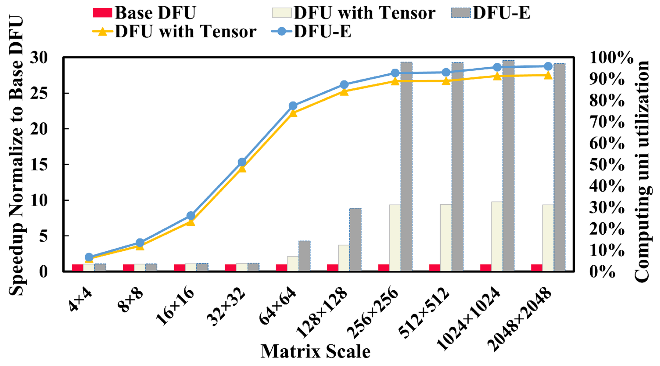


Fig. 12. Speedup and Computing unit utilization of GEMM.

flow involves multiple typical dsp kernels, as well as YOLOv5s, a commonly-used detection model with high real-time demand. These two algorithms are one of the major end-to-end benchmarks. Another benchmark is the hybrid processing of Laser Radar (Millimeter-Wave Radar) processing and CRNN (Convolutional Recurrent Neural Network) detection, which is also a representative DSP+AI workload in intelligent transportation. In addition, we also add the Transformer network to the evaluation, which is widely used in NLP and CV domains. Experiments can show the superiority of DFU-E over GPU and ASICs.

B. Scalability and Performance

1) Tensor Unit: To validate the effectiveness of our Tensor unit design, we conducted ablation experiments encompassing the following three scenarios. We tested experiments with matrix sizes ranging from 4 to 2048. Fig. 12 demonstrates our proposed methods effectiveness in terms of performance and utilization improvements. We computed speedup ratios of matrix operations at different scales for the Base DFU (without Tensor Unit), DFU (with Tensor Unit but without mapping methods) and DFU-E (with Tensor Unit and mapping method we proposed in this paper). Adding the basic Tensor computing Unit to the DFU showed an average performance improvement of 4.80×, reaching a 9.47× improvement for matrices over 256. The acceleration effect is less pronounced for small-scale matrix operations due to the minimal data size. Each PE unit utilized in our experiment can only store up to 64 × 64 matrix data, so better acceleration effects manifests from this scale upwards. Moreover, as the experiment instantiates a 4 × 4 PE array, more substantial acceleration occurs above 256. For the DFU-E, there is no significant difference from DFU with Tensor units for smaller scales (below 32) due to minimal data size failing to leverage optimized parallelism and storage architecture. Across all scales, DFU-E shows an average performance improvement of 13.49× over the Base DFU. When on-chip storage is insufficient to store the entire matrix and using the mapping algorithm we proposed in this paper, our optimized design can be fully effective, resulting in an average performance improvement of 29.34× over the Base DFU. Our newly implemented Tensor unit achieves a 9.48× performance improvement across various scales of matrix operations, and the dataflow graph mapping

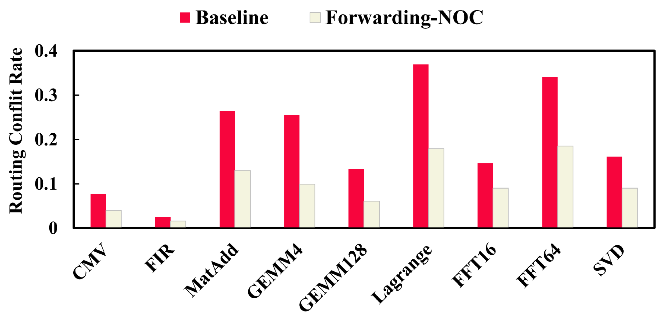


Fig. 13. Routing conflict rates of NoC.

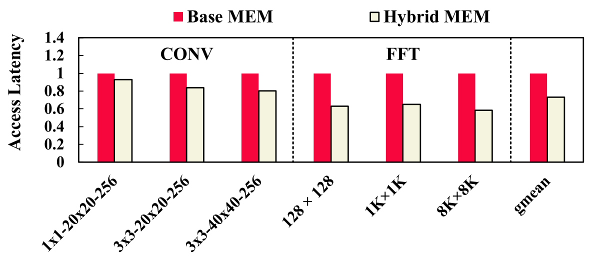


Fig. 14. Access Latency Comparison of Base MEM and Hybrid MEM.

methods we proposed in this paper yields a 3.09× performance enhancement.

2) NoC and Memory: To evaluate the effectiveness of our forwarding network, we perform conflict comparison between our design and the step-by-step mesh baseline. The comparison result is shown in Fig. 13, where routing conflict rate refers to the percentage of package conflicts in total package transfers while PE array is running. Our forwarding design substantially reduces the routing conflict rate by 41% on average. These results show that forwarding connection can alleviate or even resolve the bottleneck of network congestion of dataflow architecture.

To evaluate the effectiveness of our hybrid on-chip memory design, we implemented the following two different configurations: 1) Base MEM: It is our baseline, using the DFU-E with on-chip scratchpad memory (SPM), but without hybrid on-chip memory. 2) Hybrid MEM: It replaces the on-chip SPM with the hybrid on-chip memory. Fig. 14 shows the access latency for CONV and FFT with different data sizes using SPM and hybrid on-chip memory, normalized to SPM. Compared to SPM, Hybrid MEM effectively mitigates memory access latency, resulting in reductions of 17.1% on average. For small-scale regular applications like small-scale CONV, where the access and computation patterns are highly fixed and all data can be efficiently prefetched, both Hybrid MEM and SPM demonstrate equivalent low access latency. Conversely, in scenarios where the proportion of data with a large span of memory access addresses is not negligible, Hybrid MEM exhibits notable access latency reductions: 34.8% (1 K× 1K FFT), 41.6% (8K× 8K FFT). The performance discrepancy of SPM in large-scale applications primarily arises from its lack of the dynamic global addressing capability, which severely impacts its access efficiency.

C. Versus GPU

We compared the DFU-E with NVIDIA Jetson AGX Orin 64 GB, which is the most powerful processor for edge computing. As depicted in Table III, our DFU-E has a slightly lower performance in terms of the peak performance on FP64, FP32, FP16, and INT8 data types. Orin has a larger on-chip memory. The memory bandwidth is the same as 204.8 GB/s. In the experiment, we use single and double precision for DSP applications and half-precision for AI applications, we try to use CUDA's deep-optimized library to implement the corresponding algorithms, such as CuDNN, CuFFT, and cuBLAS. We also compared the performance and energy efficiency of neural networks at different precisions, as shown in Fig. 18. For int8, DFU-E's performance is slightly lower than Orin's because the peak computation power is lower than Orin, but DFU-E achieves higher energy efficiency.

Fig. 15 shows the advantages of DFU-E over Orin in terms of utilization and speedup. In the experiment, we used two DSP and AI core kernels of different scales, FFT and Convolution algorithms. The results show that in the small amount of data, DFU-E has a clear advantage over Orin, especially for FFT. When the data size gradually increases, the experimental results tend to be consistent. The significant performance improvements achieved by DFU-E rely on the effective enhancement of multi-level parallelism of the dataflow execution model, SPM, and DMA mechanism, as well as optimizations applied by the software stack (such as optimized compiler and high-performance libraries). On the other hand, we have deeply optimized task-initialization and task-switching efficiency, which can gain advantages for lightweight workloads. For example, FFT 128 × 128 achieves the highest utilization (4.8×) and speedup (5.6×) over Orin. When it comes to FFT 8K × 8K, the factor affecting performance is gradually changed from memory access to calculation, and the performance gap gradually narrows.

The performance and energy efficiency of DSP and AI applications are shown in Fig. 16, including SAR imaging, YOLO, CRNN, Transformer and Laser Radar. We also demonstrate the $\pm 2SD$ standard deviation of error bar in the speedup results. error bar of The detailed algorithm steps of SAR imaging and YOLOv5s are shown in Fig. 2. As we can see from the results, although the peak computing power of DFU-E is lower than that of Orin, due to the higher utilization, DFU-E achieves better performance on most of the workloads. For example, DFU-E gains up to 1.81× performance advantage on the *MotionCompensation2* stage, and 1.42× on average for SAR imaging. Other applications, DFU-E achieves 1.03×, 1.67×, 1.05×, 1.12× and 1.33× for YOLO, CRNN, Transformer-bs64, Transformer-bs512 and Laser radar, respectively. For large-scale workloads, such as *Backbone (Conv 1, 2, 4)*, *Linear6-bs64*, and *Attention-bs512*, Orin also achieves higher utilization, therefore resulting in higher performance due to higher peak computation power.

In addition to performance testing, we also measured energy efficiency, which is more reflective of the effectiveness of the design. The result reveals that, for most workloads, DFU-E has certain advantages over Orin, such as SAR imaging (1.27× on

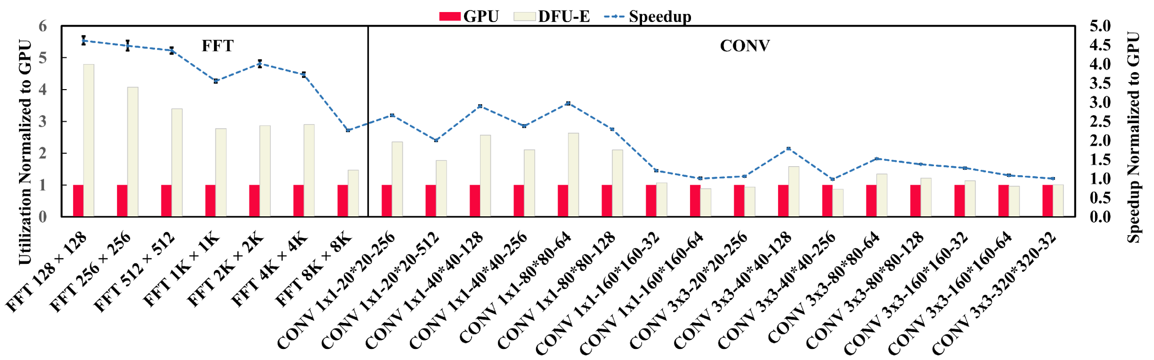


Fig. 15. Utilization and speedup of FFT and Conv with different batch size.

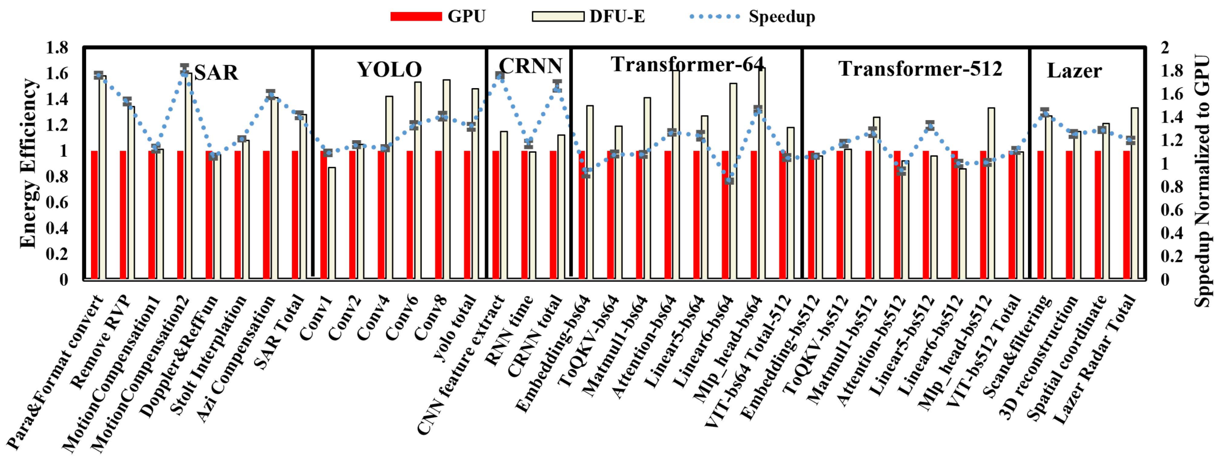


Fig. 16. Speedup and energy efficiency comparison between DFU-E and GPU.

average), CRNN (1.12× on average), Transformer-bs64 (1.18× on average) and Laser Radar (1.22× on average). For YOLO and Transformer-bs512, DFU-E is slightly less energy efficient than Orin (0.97× and 0.99× respectively), because Orin has a slight advantage of the utilization of the Convolution workloads, which accounts for the main proportion of the computation. It should be noted that in the SAR algorithm, the interpolation calculation steps (including *sinc* interpolation and *lagrange* interpolation) adopt the Cache mode because of irregular data access. Compared with SPM mode, Cache mode achieves a 6.3% performance improvement.

D. Versus ASIC

To verify the execution effectiveness of DFU-E compared to domain-specific accelerators on different AI and DSP applications, we selected MC-YOLO [42] and TI C6678 as dedicated accelerators for AI and DSP domains, respectively. TI C6678 is a prominent DSP for all the DSP applications mentioned in our experiment with 8 cores, while each core has 16-FP adders/multipliers, using DSPLIB C66x 3.4.0.0. We normalized the peak performance of accelerators to DFU-E.

Fig. 17 demonstrates the improvement in energy efficiency and speedup achieved by DFU-E compared to two domain specific accelerators. First, compared to the YOLO-specific accelerator, MC-YOLO, DFU-E achieves better energy efficiency

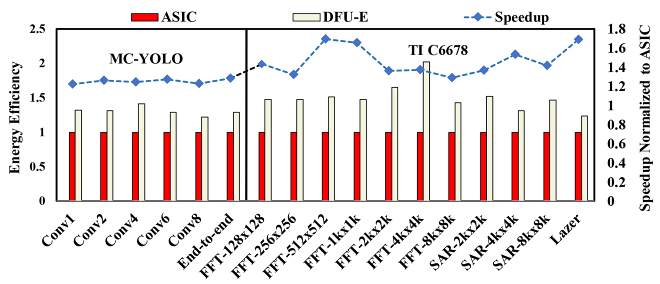


Fig. 17. Energy Efficiency and Speedup over ASICs (Normalized to ASICs).

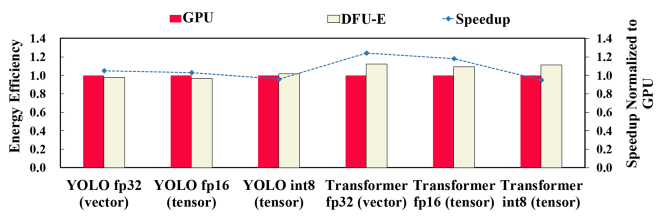


Fig. 18. Comparison of efficiency with different precision.

on each representative convolutional layer of YOLO, and exhibits nearly 1.29× of energy efficiency improvement and 1.27× of speedup in end-to-end experiments. Second, DFU-E also achieves a maximum 2.02× of energy efficiency improvement and 1.69× of speedup in different scales of DSP applications such as FFT and SAR, with an average energy efficiency improvement and speedup of 1.47× and 1.51×, respectively.

E. Discussion

The experiment results show that DFU-E (12 nm) gained better energy efficiency advantages over NVIDIA Jetson Orin (8 nm). We believe that the benefits come from deep optimization of hardware and software co-design. In terms of hardware, we adopt a novel multi-layer parallel dataflow execution model and reconfigurable PE/sPE architecture, which improves the utilization of computing resources and reduces the overhead of task scheduling and switching. This is reflected in the better performance and energy efficiency advantages of DFU-E for small-scale tasks. We also optimize NoC, on-chip memory and other modules to further accelerate memory access efficiency. An efficient and easy-to-use compiler and programming model are provided to help developers to exploit higher efficiency. All these hardware and software design decisions contribute to the higher performance and energy efficiency that DFU-E offers for edge computing.

V. Related Works

In recent years, many DSP and AI accelerators that can be applied in edge computing scenarios have been proposed to advance the performance improvement of DSP or AI applications [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53].

Jetson AGX Orin [54], released by NVIDIA, is a typical, powerful, and energy-efficient general-purpose accelerator that is mainly used for edge computing, including autonomous driving, smart city, industrial manufacturing, etc., which has high computing power for both DSP and AI applications. In this paper, we mainly conducted a comprehensive comparison with Orin in DSP and AI processing. Of course, Orin is a very comprehensive and excellent general-purpose accelerator. This paper does not focus on other functional modules in Orin.

Mozart [37] manipulates the dataflow and data reuse as fundamental architecture primitives to effectively support AI applications with low latency and high performance. Plasticine [55] (SambaNova [56]) is a novel reconfigurable architecture for efficiently executing both sparse and dense AI applications composed of parallel patterns. Certainly, there are numerous excellent architectures for AI processing, such as Groq [57], Goya [58],TPUv4 [59], DianNao series [60], [61] and dataflow architectures by Tony Nowatzki's team [20], [62], [63]. Besides, some works focus on the scheduling optimization of AI applications on edge devices. HiTDL [64] discusses the challenges of efficiently deploying deep neural networks in mobile edge environments, and it proposes the HiTDL framework and models the performance of DNN inference latency and throughput. [65], [66] study the task allocation problem when performing multi-task migrated learning (MTL) on resource-constrained edge devices and propose a data-driven collaborative task allocation (DCTA) approach, while TrimCaching [67] proposes a novel model placement scheme called Parameter Shared Model Caching (TrimCaching) for caching AI models in wireless edge networks. All these genius innovations provide us with a lot of design inspiration.

For DSP applications, due to the diversity of DSP algorithms, designers usually implement high-width vector units based on general-purpose CPUs or FPGAs to accelerate DSP applications, which has relatively low efficiency due to the inevitable instruction loading and parsing process [68], and are insufficiently adaptable to the fast development of DSP algorithms. Texas Instruments [69] develops a series of multicore DSPs and implements complete DSP operator libraries, which are widely used in the world. RASP [70] uses the associated reconfigurable spatial architectures to further resolve the problem, among which dataflow theories are widely used for fully exploiting the data parallelism [32]. Compared to them, our DFU-E improves execution parallelism and reduces memory access based on the dataflow execution model, while taking into account vector computing efficiency and general computing capabilities.

VI. Conclusion

There is no doubt that edge computing has become one of the most important computing scenarios. The design of edge processor architecture with high computing power, high energy efficiency, and strong versatility has become a current research hotspot, which is also our concern in this paper. Therefore, we propose a dataflow-based reconfigurable architecture equipped with a novel compute engine, memory hierarchy, NoC, power management, and optimized programming and compilation software stack. The multi-layer parallel execution model supported by DFU-E fully utilizes hardware resources and improves energy efficiency. The results underscore the effectiveness of DFU-E's design, which emphasizes performance, efficiency, and flexibility in the ever-evolving landscape of computing systems, particularly within the context of edge computing.

Looking ahead, we plan to enhance our architecture through adaptive dataflow graph optimization, heterogeneous computing support, and scalable distributed scheduling. By developing adaptive dataflow architectures that modify graph structures dynamically based on real-time workloads, we aim to create self-optimizing systems supported by machine learning techniques like reinforcement and online learning. Additionally, integrating heterogeneous computing will enable our architecture to flexibly leverage CPUs, GPUs, and specialized accelerators within a unified dataflow model, ensuring efficient adaptation to diverse computational needs. Finally, advancing distributed scheduling techniques will allow us to manage concurrent dataflow graphs across edge devices in a scalable and reliable manner, minimizing control bottlenecks.

Acknowledgment

The authors sincerely thank Shengzhong Tang, Yuhang Qiu, Zhiyuan Zhang, Yudong Mu and Shantian Qin for their invaluable contributions to chip development and the experimental work presented in this paper. Their expertise and dedication have been instrumental in advancing this research

References

[1] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[2] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020.

[3] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," in *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, Aug. 2019.

[4] L. Chen, M. Odema, and M. A. A. Faruque, "Romanus: Robust task offloading in modular multi-sensor autonomous driving systems," in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Diego, California, USA, 2022, pp. 162:1–162:8, doi: 10.1145/3508352.3549356.

[5] C. Hao et al., "NAIS: Neural architecture and implementation search and its applications in autonomous driving," in *Proc. Int. Conf. Comput.-Aided Des.*, Westminster, CO, USA, 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICCAD45719.2019.8942055

[6] W. Shi, M. B. Alawieh, X. Li, H. Yu, N. Aréchiga, and N. Tomatsu, "Efficient statistical validation of machine learning systems for autonomous driving," in *Proc. 35th Int. Conf. Comput.-Aided Des.*, Austin, TX, USA, 2016, Art. no. 36. doi: 10.1145/2966986.2980077.

[7] P. Arthurs, L. Gillam, P. Krause, N. Wang, K. Halder, and A. Mouzakitis, "A taxonomy and survey of edge cloud computing for intelligent transportation systems and connected vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6206–6221, Jul. 2022.

[8] Q. Chen, A. K. Sowan, and S. Xu, "A safety and security architecture for reducing accidents in intelligent transportation systems," in *Proc. Int. Conf. Comput.-Aided Des.*, San Diego, CA, USA, 2018, Art. no. 95, doi: 10.1145/3240765.3243462.

[9] B. Zheng, M. O. Sayin, C. Lin, S. Shiraishi, and Q. Zhu, "Timing and security analysis of vanet-based intelligent transportation systems: (invited paper)," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Irvine, CA, USA, 2017, pp. 984–991, doi: 10.1109/ICCAD.2017.8203888.

[10] L. Dong, X. Meng, and D. Zhu, "High-squint SAR imaging technique based on multi-chip DSP," in *Proc. 7th Asia-Pacific Conf. Synthetic Aperture Radar*, 2021, pp. 1–5.

[11] Z. Liu et al., "Robust target recognition and tracking of self-driving cars with radar and camera information fusion under severe weather conditions," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6640–6653, Jul. 2022.

[12] D. Kirk et al., "NVIDIA CUDA software and GPU parallel computing architecture," in *Proc. 6th Int. Symp. Memory Manage.*, 2007, pp. 103–104.

[13] V. Leon, K. Pekmestzi, and D. Soudris, "Exploiting the potential of approximate arithmetic in DSP & AI hardware accelerators," in *Proc. IEEE 31st Int. Conf. Field- Program. Log. Appl.*, 2021, pp. 263–264.

[14] S. Zhao, F. Blaabjerg, and H. Wang, "An overview of artificial intelligence applications for power electronics," *IEEE Trans. Power Electron.*, vol. 36, no. 4, pp. 4633–4658, Apr. 2021.

[15] G. Furano et al., "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 35, no. 12, pp. 44–56, Dec. 2020.

[16] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, Mar./Apr. 2008.

[17] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2018, pp. 522–531.

[18] S. Mittal, "A survey on optimized implementation of deep learning models on the Nvidia Jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, 2019.

[19] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar./Apr. 2021.

[20] V. Dadu, J. Weng, S. Liu, and T. Nowatzki, "Towards general purpose acceleration by exploiting common data-dependence forms," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2019, pp. 924–939.

[21] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[22] H.-N. Yu, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974.

[23] R. Machupalli, M. Hossain, and M. Mandal, "Review of asic accelerators for deep neural network," *Microprocessors Microsystems*, vol. 89, 2022, Art. no. 104441.

[24] W. Wu et al., "Application of local fully convolutional neural network combined with YOLO V5 algorithm in small target detection of remote sensing image," *PLoS One*, vol. 16, no. 10, 2021, Art. no. e0259283.

[25] M. Soumekh, *Synthetic Aperture Radar Signal Processing*, vol. 7. New York, NY,USA: Wiley, 1999.

[26] D. Bhatt et al., "CNN variants for computer vision: History, architecture, application, challenges and future scope," *Electronics*, vol. 10, no. 20, 2021, Art. no. 2470.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2012, pp. 84–90.

[28] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[29] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. 13th Eur. Conf. Comput. Vis.*, Zurich, Switzerland, 2014, pp. 818–833. [Online]. Available: https://doi.org/10.1007/978-3-319-10590-1_53

[30] A. Fuchs and D. Wentzlaff, "The accelerator wall: Limits of chip specialization," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 1–14.

[31] J. B. Dennis, "First version of a data flow procedure language," in *Proc. Program. Symp. Colloque La Programmation*, Paris, France, 1974, pp. 362–376. [Online]. Available: https://doi.org/10.1007/3-540-06859-7_145

[32] Arvind and R. S. Nikhil, "Executing a program on the MIT tagged-token dataflow architecture," *IEEE Trans. Comput.*, vol. 39, no. 3, pp. 300–318, Mar. 1990. [Online]. Available: https://doi.org/10.1109/12.48862

[33] K. Sankaralingam et al., "TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 62–93, 2004, doi: 10.1145/980152.980156.

[34] D. Burger et al., "Scaling to the end of silicon with edge architectures," *Computer*, vol. 37, no. 7, pp. 44–55, Jul. 2004, doi: 10.1109/MC.2004.65.

[35] J. Weng, S. Liu, Z. Wang, V. Dadu, and T. Nowatzki, "A hybrid systolic-dataflow architecture for inductive matrix algorithms," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 703–716.

[36] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "Wavescalar," in *Proc. 36th Annu. Int. Symp. Microarchit.*, San Diego, CA, USA, 2003, pp. 291–302, doi: 10.1109/MICRO.2003.1253203.

[37] K. Sankaralingam et al., "The mozart reuse exposed dataflow processor for AI and beyond: Industrial product," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 978–992.

[38] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, 2016.

[39] A. Li, G.-J. van den Braak, H. Corporaal, and A. Kumar, "Fine-grained synchronizations and dataflow programming on gpus," in *Proc. 29th ACM Int. Conf. Supercomputing*, 2015, pp. 109–118.

[40] G. G. J. Suettlerlein and S. Zuckerman, "An implementation of the codelet model," in *Euro-Par Parallel Processing.*, ser. Lecture Notes in Computer Science, vol. 8097. Berlin, Germany: Springer, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-40047-6_63

[41] X. Ye, D. Fan, N. Sun, S. Tang, M. Zhang, and H. Zhang, "Simict: A fast and flexible framework for performance and power evaluation of large-scale architecture," in *Proc. Int. Symp. Low Power Electron. Des.*, 2013, pp. 273–278.

[42] T. Adiono, R. M. Ramadhan, N. Sutisna, I. Syafalni, R. Mulyawan, and C.-H. Lin, "Fast and scalable multicore YOLOv3-tiny accelerator using input stationary systolic architecture," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 31, no. 11, pp. 1774–1787, Nov. 2023.

[43] V. Leon, T. Paparouni, E. Petrongonas, D. Soudris, and K. Z. Pekmestzi, "Improving power of DSP and CNN hardware accelerators using approximate floating-point multipliers," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, pp. 39:1–39:21, 2021, doi: 10.1145/3448980.

[44] R. Prasad, S. Das, K. J. M. Martin, and P. Coussy, "Floating point CGRA based ultra-low power DSP accelerator," *J. Signal Process. Syst.*, vol. 93, no. 10, pp. 1159–1171, 2021, doi: 10.1007/s11265-020-01630-2.

[45] K. Tsoumanis, S. Xydis, G. Zervakis, and K. Z. Pekmestzi, "Flexible DSP accelerator architecture exploiting carry-save arithmetic," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 1, pp. 368–372, Jan. 2016, doi: 10.1109/TVLSI.2015.2390974.

[46] Y. Fu et al., "A DSP-purposed reconfigurable acceleration machine (DREAM) for high energy efficiency MIMO signal processing," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 70, no. 2, pp. 952–965, Feb. 2023, doi: 10.1109/TCSI.2022.3220947.

[47] H. Liao et al., "Ascend: A scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture*, Seoul, South Korea, 2021, pp. 789–801, doi: 10.1109/HPCA51647.2021.00071.
[48] O. Wechsler, M. Behar, and B. Daga, "Spring hill (NNP-I 1000) intel's data center inference chip," in *Proc. IEEE Hot Chips 31 Symp.*, 2019, pp. 1–12.
[49] S. Bavikadi et al., "A survey on machine learning accelerators and evolutionary hardware platforms," *IEEE Des. Test*, vol. 39, no. 3, pp. 91–116, Jun. 2022. [Online]. Available: https://doi.org/10.1109/MDAT.2022.3161126
[50] C. Eckert et al., "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *IEEE Micro*, vol. 39, no. 3, pp. 11–19, May/Jun. 2019, doi: 10.1109/MM.2019.2908101.
[51] Y. Jiao, L. Han, and X. Long, "Hanguang 800 NPU - the ultimate AI inference solution for data centers," in *Proc. IEEE Hot Chips 32 Symp.*, Palo Alto, CA, USA, 2020, pp. 1–29, doi: 10.1109/HCS49909.2020.9220619.
[52] X. Hu, X. Li, H. Huang, X. Zheng, and X. Xiong, "TiNNA: A tiny accelerator for neural networks with efficient DSP optimization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 4, pp. 2301–2305, Apr. 2022, doi: 10.1109/TCSII.2022.3150980.
[53] D. Wang, K. Xu, J. Guo, and S. Ghiasi, "DSP-efficient hardware acceleration of convolutional neural network inference on FPGAs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4867–4880, Dec. 2020, doi: 10.1109/TCAD.2020.2968023.
[54] Nvidia, NVIDIAJetsonAGX Orin (DS-10662-001_v1.1), 2022.
[55] R. Prabhakar et al., "Plasticine: A reconfigurable architecture for parallel patterns," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 389–402.
[56] R. Prabhakar, S. Jairath, and J. L. Shin, "Sambanova SN10 RDU: A 7 nm dataflow architecture to accelerate software 2.0," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2022, pp. 350–352.
[57] G. Dale Southard, Ecosystem solutions distinguished architect, "Whitepaper: Tensor streaming architecture delivers unmatched performance for compute-intensive workloads," 2019. [Online]. Available: https://groq.com/wp-content/uploads/2019/10/Groq_Whitepaper_2019Oct.pdf
[58] E. Medina and E. Dagan, "Habana labs purpose-built AI inference and training processor architectures: Scaling AI training systems using standard ethernet with gaudi processor," *IEEE Micro*, vol. 40, no. 2, pp. 17–24, Mar./Apr. 2020.
[59] N. P. Jouppi et al., "Ten lessons from three generations shaped Google's TPUv4i : Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 1–14.
[60] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Feb. 2014. [Online]. Available: https://doi.org/10.1145/2654822.2541967
[61] D. Liu et al., "PuDianNao: A polyvalent machine learning accelerator," in *Proc. 20th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, 2015, pp. 369–381, doi: 10.1145/2694344.2694358.
[62] K. Sankaralingam et al., "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," *IEEE Micro*, vol. 23, no. 6, pp. 46–51, Nov./Dec. 2003, doi: 10.1109/MM.2003.1261386.
[63] V. Govindaraju et al., "DySER: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, Sep./Oct. 2012, doi: 10.1109/MM.2012.51.
[64] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, "HiTDL: High-throughput deep learning inference at the hybrid mobile edge," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4499–4514, Dec. 2022.
[65] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "On-edge multi-task transfer learning: Model and practice with data-driven task allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1357–1371, Jun. 2020.
[66] X. Wang, Y. Niu, F. Liu, and Z. Xu, "When FPGA meets cloud: A first look at performance," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1344–1357, Apr.-Jun. 2022.
[67] G. Qu, Z. Lin, F. Liu, X. Chen, and K. Huang, "Trimcaching: Parameter-sharing AI model caching in wireless edge networks," in *Proc. IEEE 44th Int. Conf. Distrib. Comput. Syst.*, 2024, pp. 36–46.
[68] L. Shannon, "Reconfigurable computing architectures," *Reconfigurable Comput.*, 2008.
[69] D. Wang and M. Ali, "Synthetic aperture radar on low power multi-core digital signal processor," in *Proc. IEEE Conf. High Perform. Extreme Comput.*, Waltham, MA, USA, 2012, pp. 1–6, doi: 10.1109/HPEC.2012.6408665.
[70] F. Feng, L. Li, K. Wang, F. Han, B. Zhang, and G. He, "Floating-point operation based reconfigurable architecture for radar processing," *IEICE Electron. Exp.*, vol. 13, no. 21, Art. no. 20160893, 2016, doi: 10.1587/elex.13.20160893.

Wenming Li received the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2016. He is currently an associate professor in Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include high-throughput processor architecture, dataflow architecture, and software simulation.

Zhihua Fan received the BS degree from JiLin University, in 2018 and the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2024. He is currently an assistant professor with the Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include dataflow architecture, programming model, and reconfigurable architecture.

Tianyu Liu received the BS degree from Southeast University, Nanjing, China, in 2020. She is currently working toward the post-graduate with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Her current research interests include high- throughput computer architecture and dataflow architecture.

Zhen Wang received the BS degree from the University of Chinese Academy of Sciences, Beijing, China, in 2021. She is currently working toward the post-graduate with the Institute of Computing Technology, Chinese Academy of Sciences. Her current research interests include high-throughtput computer architecture and dataflow architecture.

Haibin Wu received the BS degree from the Huazhong University of Science and Technology, China, in 2015. He is currently working toward the PhD degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His main research interests include computer architecture, signal processing, and machine learning algorithm.

Meng Wu received the MS degree from the University of Chinese Academy of Sciences, Beijing, China, in 2017. She is currently an engineer with the Institue of Computing Technology, Chinese Academy of Sciences. Her current research interests include dataflow architecture and high-throughtput computer architecture.

Kunming Zhang received the MS degree from the Beijing Institute of Technology. He is currently working toward the PhD degree with the Institute of Computing Technology of the Chinese Academy of Sciences. His current research interests include dataflow architecture and high-throughtput computer architecture.

Yanhuan Liu received the BS degree in electronic science and technology from the Beijing University of Posts and Telecommunications, in 2015 and the MS degree in integrated circuit engineering from Tsinghua University, in 2018. He is currently working toward the PhD degree with the Institute of Computing Technology of the Chinese Academy of Sciences. His main research interests include high-performance dataflow architecture and high-throughput computer architecture.

Ninghui Sun received the BS degree from Peking University, in 1989, and the MS and PhD degrees in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 1992 and 1999, respectively. He is a professor with the Institute of Computing Technology, Chinese Academy of Sciences. He is the architect of the Dawning series high-performance computers. His research interests include computer architecture, operating system, and parallel algorithm.

Xiaochun Ye received the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2010. He is currently a professor in Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include high-performance computer architecture and software simulation.

Dongrui Fan received the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2005. He is currently a professor and PhD supervisor in Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include high-throughput computer architecture and high- performance computer architecture.