Back to Homogeneous Computing: A Tightly-Coupled Neuromorphic Processor With Neuromorphic ISA

Zhijie Yang, Lei Wang, Wei Shi, Yao Wang, Junbo Tie, Feng Wang, Xiang Yu, Linghui Peng, Chao Xiao, Xun Xiao, Yao Yao, Gan Zhou, Xuhu Yu, Rui Gong, Xia Zhao, *Member, IEEE*, Yuhua Tang, and Weixia Xu

***Abstract*—In recent years, neuromorphic processors are widely used in many scenarios, showing extreme energy efficiency over traditional architectures. However, almost all existing neuromorphic hardware are following the heterogeneous computing methodology without Instruction Set Architecture (ISA), leading to inflexibility in programming. In this paper, we first propose a RISC-V Neuromorphic Extension (RVNE) to enable fine-grained and flexible homogeneous programming for neuromorphic algorithms while utilizing SNN sparsity from different levels of granularity and computing flows. Based on RVNE, we next implement a neuromorphic micro-architecture that is tightly coupled to the CPU pipeline to accelerate neuromorphic computing. To demonstrate the proposed homogeneous neuromorphic architecture, we implement a prototype processor called NeuroRVcore based on RISC-V ISA and an open-source RISC-V core. The evaluation results show that RVNE achieves a 2.8 × −4.3 × reduction in code density compared with the general-purpose ISAs. Compared with the state-of-the-art neuromorphic processor, the proposed homogeneous computing reduces energy consumption by 3.4%−22.5% while enabling fine-grained and flexible homogeneous programming.**

***Index Terms*—Instruction set architecture, neuromorphic computing, neuromorphic processor, RISC-V, spiking neural network.**

Manuscript received 21 October 2022; revised 16 August 2023; accepted 17 August 2023. Date of publication 22 August 2023; date of current version 15 September 2023. This work was supported in part by the National Natural Science Foundation of China under Grants 62372461, 62032001, 62203457, and 62102438, and in part by the Key Laboratory of Advanced Microprocessor Chips and Systems. Recommended for acceptance by A. C. Zhou. *(Corresponding author: Lei Wang.)*

Zhijie Yang, Wei Shi, Yao Wang, Junbo Tie, Xiang Yu, Linghui Peng, Chao Xiao, Xun Xiao, Gan Zhou, Xuhu Yu, Rui Gong, Yuhua Tang, and Weixia Xu are with the College of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan 410073, China (e-mail: yangzhijie@nudt.edu.cn; shiwei@nudt.edu.cn; wangyaobsz@nudt.edu.cn; tiejunbo11@nudt.edu.cn; ricardoyx@foxmail.com; penglinghui1997@163.com; xiaochao@nudt.edu.cn; xiaoxun520@nudt.edu.cn; 1252691981@qq.com; yxh_2016@nudt.edu.cn; rgong@nudt.edu.cn; yhtang62@163.com; xuweixia@nudt.edu.cn).

Lei Wang is with the Defense Innovation Institute, Academy of Military Sciences, Beijing 100850, China, and also with the College of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan 410073, China (e-mail: leiwang@nudt.edu.cn).

Feng Wang and Yao Yao are with the College of Computer Science, Electronic Engineering, Hunan University, Changsha, Hunan 410012, China (e-mail: wangfeng@hnu.edu.cn; yaoyao21@hnu.edu.cn).

Xia Zhao is with the Defense Innovation Institute, Academy of Military Sciences, Beijing 100850, China (e-mail: zhaoxiahust@gmail.com).

Digital Object Identifier 10.1109/TPDS.2023.3307408

I. Introduction

NEUROMORPHIC computing is a type of computing technology that mimics the information-processing principles and rules of the human biological brain. In a narrow sense, it is specifically referred to as computations and algorithms based on spiking neural networks (SNNs) at present and not limited to SNNs in the future. Compared with the traditional deep neural networks (DNNs), SNNs are more bio-mimetic and work in an event-driven manner rather than the manner that responds to all inputs like DNNs. Besides, SNNs take spikes as signal transmissions between neurons rather than continuous values like DNNs. Thus, SNN is a promising solution to build more energy-efficient systems than DNN [1].

In recent years, neuromorphic processors [2], [3], [4], which serve as the running platforms of neuromorphic applications mainly represented by SNN algorithms, have emerged continuously. Represented by TrueNorth [5] of IBM and Loihi 1/2 [6] of Intel, Neuromorphic processors are already used in a wild range of fields such as robot control [7], unmanned autonomous system decision-making [8], Brian Machine Interface (BMI) signal processing [9], and other applications, showing impressive energy efficiency advantages compared with CPUs, GPUs, and DNN accelerators.

However, current neuromorphic processors all follow a roadmap without Instruction Set Architecture (ISA). Compared to them, exploiting homogeneous computing with ISA to accelerate neuromorphic computing can further bring better programmability because the homogeneous neuromorphic architecture provides sufficient headroom for upper-level compilation optimization by enabling finer-grain instruction-level parallelism. With this advantage, homogeneous computing with ISA is a promising solution to promote the development of neuromorphic computing ecology by decoupling the neuromorphic software and hardware. Recent examples include the matrix math accelerators in Power10 [10], ARM SME [11] and the Dojo chip [12].

Unfortunately, how to accelerate neuromorphic computing based on homogeneous computing has never been explored before. In this paper, we present the first attempt, to our best knowledge, to propose a homogeneous architecture-level solution for neuromorphic computing. To achieve that, two challenges have to be addressed, i.e., i) designing an ISA that conforms to the characteristics of SNNs for homogeneous computing, and ii) implementing an energy-efficient homogeneous micro-architecture in the synchronous circuit to efficiently support the event-driven features.

In response to the above challenges, the following aspects have to be fully investigated. First, a list of the most common operators has to be extracted from the traces and profiles of representative SNNs with a wide coverage of network structures and input datasets, and then the instruction level of parallelism has also to be obtained from the dynamic sparsity analysis. Second, an efficient instruction set has to be defined with different parallelism granularities according to the profiling & analyzing results, supporting the whole execution process of SNNs while balancing the trade-off between the hardware overhead and software programming flexibility. Further, the micro-architectural level implementation of the proposed homogeneous neuromorphic ISA has to be demonstrated, for which tight-coupling of the neuromorphic computing elements with a general-purpose processor's pipeline are required since both the processing capability of event-driven data and a unified memory access scheme with CPU have to be accommodated on the same implementation.

The main contributions can be summarized as follows:

- Compared to widely-used heterogeneous neuromorphic architectures, we present the first attempt, to our best knowledge, to propose a homogeneous architecture-level solution for neuromorphic computing.
- We have proposed a neuromorphic instruction set named "RVNE" aiming at homogeneous neuromorphic computing to fully exploit the sparsity features of neuromorphic applications for the first time in academia and industry.
- We have proposed a neuromorphic micro-architecture for RVNE's implementation by modifying a classical general-purpose instruction pipeline, to accelerate the SNN operations.
- We have fully demonstrated a homogeneous neuromorphic prototype processor named "NeuroRVcore", based on RISC-V ISA and an open-source RISC-V core named "RI5CY" [13].

The evaluation results show that our homogeneous design achieves a 2.8 × −4.3 × reduction in code density compared with the general-purpose ISAs and brings a 3.4%−22.5% reduction in energy consumption compared with the state-of-the-art neuromorphic processors.

II. Background and Motivation

A. Neuromorphic Processor

Neuromorphic processors are non-Von-Neumann computing systems and the applications running on them are neuromorphic algorithms mainly represented by SNNs at present but not limited to SNNs in the future. Many well-known neuromorphic processors are emerging in recent years, such as IBM's TrueNorth [5], Intel's Loihi [6], INI's DyNaps [2], Stanford University's Neurogird [4], SpiNNaker of the University of Manchester [3], etc. Neuromorphic processors show extreme energy efficiency over traditional architectures and are

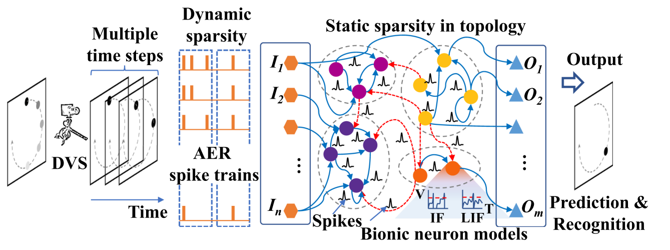


Fig. 1. Schematic diagram of a recurrent SNN taking spike trains of the emerging sensor as input.

promising solutions to critical user cases such as autonomous vehicles [14], unmanned system decision-making [8], and other Internet of Things (IoTs) applications in the future.

Intrinsically, asynchronous circuits are more in line with the event-driven characteristics of SNN and greatly reduce system power consumption. Mainstream neuromorphic processors such as TrueNorth [5] and Loihi 1/2 [6] are all taking the asynchronous circuits as the implementation methods. Besides, almost all of the existing neuromorphic processors are generally working as co-processors of the CPU, in the form of a heterogeneous computing system.

B. Spiking Neural Network

Fig. 1 shows the structure diagram of a recurrent SNN (Liquid State Machine, LSM), which is executed in time steps. Inputs of multiple time steps generated from Dynamic Vision Sensor (DVS) or converted using Poisson encoding from RGB samples enter the network in the form of address-event representation (AER) spike trains. Spikes are used as information transferred between neurons in the network. Each neuron follows a specific bionic computational model, such as the integrate-and-fire (IF) model [15], the leaky-integrate-fire (LIF) model [16], and so on.

Taking the LIF neuron model as an example, the computational flow of neurons in the process of SNN execution mainly includes three steps: i) Neuron current accumulation: Each neuron receives input spikes from other neurons in both the input layer and hidden layer and accumulates the weight of the synaptic connection to its current value as shown in the first equation of (1); ii) Computing and leakage of voltage: Based on the current, the voltage of each neuron is calculated and the leakage of it is performed as shown in the last equation of (1); iii) Spike generation: If the voltage exceeds the pre-set threshold, an output spike will be triggered and all states of a neuron will be reset to the initial values.

$$\begin{cases} I(t) = \sum_{n=0}^{N} S_n * W_n \\ rnd\_I = I(t-1) >> I\_leaky \\ rnd\_V = V(t-1) >> V\_leaky \\ V(t) = V(t-1) - rnd\_V + I(t) - rnd\_I \end{cases} \tag{1}$$

where $I(t)$ and $V(t)$ are the current and voltage at the time step t respectively, V_leaky and I_leaky are leaky intermediate states of voltage and current respectively. n represents all fan-ins of the neuron. S_n and W_n are the received input spike and the weight corresponding to fan-in synapse n respectively.

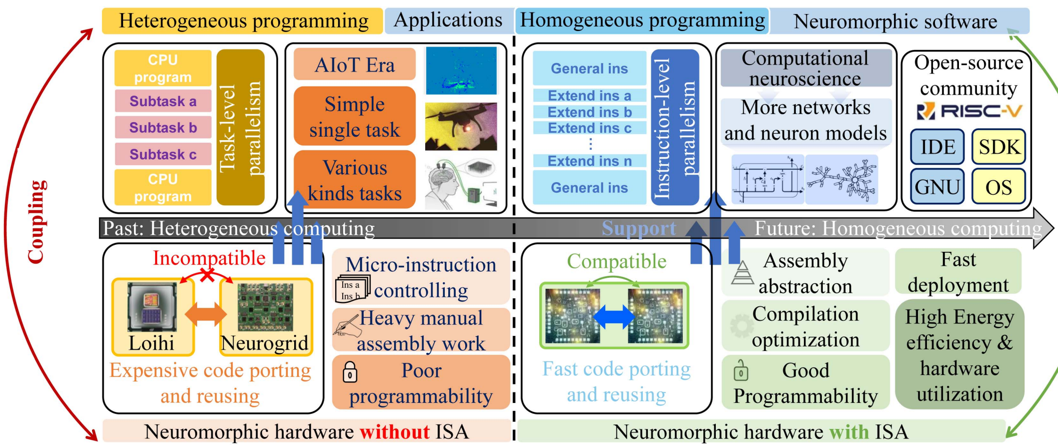


Fig. 2. The development of neuromorphic computing ecology, from architecture without ISA to obeying ISA.

C. Challenges and Opportunities

As shown in the left part of Fig. 2, nowadays, well-known neuromorphic processors such as TrueNorth [5], Loihi [6], etc. are heterogeneous computing systems without ISA that work with the assistance of the CPU, leading to difficult heterogeneous programming. For example, Loihi consists of 3 x86 CPUs and 128 neuromorphic cores. Neurogird [4] has no instruction sets and can only use micro instructions to control the processor without the help of compilers, leading to heavy manual work. Such poor programmability and incompatibility among different neuromorphic hardware without uniform ISA and SNN algorithms written in heterogeneous programming models involving different high-level languages could block the fast development of neuromorphic computing ecology, especially in future AIoT (AI+IoT) Era with various kinds of edge tasks and application scenarios.

As shown in the right part of Fig. 2, the homogeneous computing architecture conforming to a unified instruction set architecture specification is a promising solution to future neuromorphic computing development. With the same instruction set, homogeneous computing decouples the developments of neuromorphic software and hardware while maintaining their independence, ensuring full-stack compatibility (especially for high-level computing models), good programmability, and high energy efficiency due to fine-grained instruction-level parallelism from the perspective of whole programs, thus prompting the fast and sustainable development of neuromorphic computing ecology with continuously increasing neural network models and neuroscience which is tending towards more complex.

Although homogeneous computing seems to be a promising direction, how to accelerate neuromorphic computation based on homogeneous computing has never been explored before since research on neuromorphic architecture is currently in the early stages. In this paper, as shown in Fig. 3, two challenges related to the homogeneous neuromorphic design have been particularly addressed and fully investigated, i.e., designing an ISA that conforms to the characteristics of SNNs for homogeneous computing and implementing an energy-efficient homogeneous micro-architecture hardware implementation in the synchronous circuit to effectively exploit the event-driven feature of SNN.

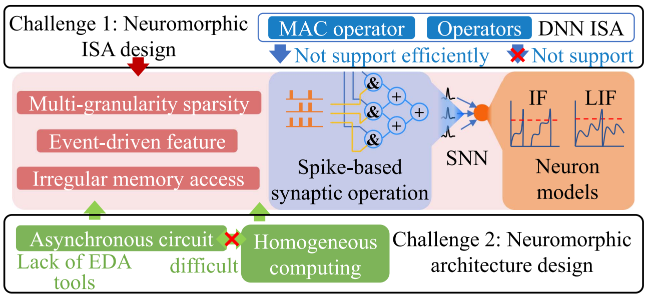


Fig. 3. The challenges of implementing homogeneous neuromorphic computing.

On one hand, as shown in Fig. 3, different from DNN, the SNN algorithm has multi-granularity sparsity (network size and topology are variable and input spike trains and synaptic connections are sparse), event-driven characteristics (neurons only work when they receive input spikes, otherwise they do not work), and irregular memory access characteristics brought by the first two characteristics to neuromorphic microarchitecture implementation. Thus, for the first challenge, although there have been many efforts to support and accelerate DNN either by customized ISA [17], [18], [19], [20] or by extended DNN instructions based on general ISA [21], [21], all of the previous works cannot be applied to accommodate SNN for that DNN ISA cannot support the complex neuron models of SNN and process address-event-representation (AER) data efficiently to exploit multi-granularity sparsity of SNNs.

On the other hand, the event-driven computing and irregular memory access characteristics brought by the sparsity of SNN increase the difficulty of microarchitecture implementation. Thus, for the second challenge, although the existing mainstream neuromorphic processors such as TrueNorth and Loihi are implemented with asynchronous circuits that are more in line with the event-driven characteristics of SNNs, compared to implementing asynchronous accelerators, implementing asynchronous homogeneous neuromorphic processors is much more difficult due to the lack of mature EDA tools for asynchronous circuit implementation and validation.

Next, we discuss how we solve these two challenges by proposing a homogeneous architecture-level solution for neuromorphic computing.

III. RVNE

In this section, the design principles of our neuromorphic ISA, representative SNN applications profiling guiding ISA design, ISA specification, and example programs will be introduced.

A. ISA Design Principles

The design principles of our neuromorphic instruction set are completeness, flexibility, efficiency, scalability, and compatibility for SNNs.

Completeness refers to the ability of the instruction set to fully cover the basic computing functions supported by various SNN application programming. Therefore, analysis of representative SNN types and topologies, as well as corresponding datasets are required for the instruction set design. This design principle is reflected in Section III-B with representative SNN application profiling. Most importantly, extending ISA based on the general instruction set guarantees the strict completeness of the ISA.

Flexibility refers to that an instruction set has instructions with different computational granularity and dataflow, providing support for upper-level compilation optimization and user programming to fully exploit the sparsity of SNN, as well as programming for SNN networks with different scales. Therefore, it is necessary to reasonably compromise the relationship between the parallelism of instruction computation and network sparsity. This design principle is reflected in Section III-B-2 with dynamic sparsity analysis of running representative SNN applications and Section III-C-2 with ISA design.

Efficiency includes two aspects, one of them is the code efficiency, and the other is execution efficiency. To achieve high code efficiency, it is necessary to analyze the computing process of SNN, combine different calculation formulas and specific functions in the calculation process, and reasonably abstract different instructions for upper-level compilation and programming use, achieving the goal of reducing code capacity and improving information density.

To improve the execution efficiency of the microarchitecture, it is necessary to analyze the bottleneck of computing and memory access and the imbalance between computing and memory access. Considering this, in the design of the instruction set architecture, designers should fully compromise the size of extended registers, computing parallelism, memory access bandwidth, and other parameters related to computing/memory access to achieve a reasonable balance. This design principle is reflected in Section III-B-1 with an analysis of the execution time breakdown of key operators in SNN applications and Section V-C-4 with design space exploration including architecture-visible extended registers defined in ISA to achieve optimal implementation efficiency.

Extensibility refers to the fact that the instruction set should be easy to expand so that new instructions and functions can be added with the development of the neuromorphic computing field. This design principle is reflected in Section VI-A ISA extensibility.

Compatibility refers to the fact that ISA should be easy to achieve compatibility in both software and hardware now and in the future. This design principle is reflected in that RVNE is extended on RISC-V ISA and NeuroRVCore is extended on open-source RISC-V core to gain the support such as toolchain and compiler from existing RISC-V ecology and community.

B. Application Profiling

To provide guidance for our neuromorphic ISA design, representative SNN applications need to be profiled. We then show our profiling method and results in this section.

The application profiling is performed in two aspects. First, to find the frequent and most time-consuming operators needed to be supported in the ISA and further be accelerated, the profiling of operation execution time breakdown is needed. Second, to find the suitable parallelism granularity of the instructions defined in the ISA, the profiling of applications' sparsity needs to be conducted.

Moreover, to make the profiling cover as many SNN structures as possible, the analyzed networks need to include typical network structures, namely convolution, recurrent, fully connected and batch normalization structures. Among them, the fully connected structure refers that any neuron in one layer of the neural network is connected with all neurons in the next layer. The convolution structure is a subset of the fully connected structure, which allows any neuron to connect with only some neurons in the latter layer. The recurrent structure allows neurons to have connections to themselves. The batch normalization layer prevents neural network overfitting and plays an important role in increasing the accuracy of the neural network.

Thus, we profile the execution trace of running DVS128 hand gesture dataset [22], LRW lipreading dataset [23] on the LSM which is a recurrent SNN and running CIFAR-10 [24] on two SCNNs which are converted from VGG-11 [25] and MobileNet V1 [26]. Among the SNNs we use, LSM contains recurrent structure, spiking VGG contains the convolutional structure and fully-connected structure, and spiking MobileNet contains batch normalization structure.

The three chosen datasets are widely used or representative. The DVS128 dataset is proposed by IBM and recommended by Mike Davies, the chief architect of Loihi, as one of the standard SNN datasets in the future [27]. It has been used in a wide range of neuromorphic processor testing such as Loihi [28] and other neuromorphic solutions [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40]. The CIFAR-10 dataset is a classic data set widely used in the traditional deep learning field. It is generally used to test the performance of SNN converted from DNN/ANN. Therefore, using this dataset is conducive to achieving the performance comparison between SNN and DNN/ANN. It has also been used in a wide range of neuromorphic processor testing such as Loihi [41] and other neuromorphic processors [42], [43], [44]. LRW dataset represents an emerging direction of SNN datasets, namely multi-modality recognition. Similarly, Ceolini et al. [45] generated multi-modal

TABLE I
THE AVERAGE EXECUTION TIME BREAKDOWN OF MAIN OPERATORS

	DVS128 LSM	LRW LSM	CIFAR-10 SVGG	CIFAR-10 SMOB
Current accumulation	29.25%	29.47%	19.33%	11.88%
Neuron computation	7.38%	10.17%	21.70%	36.21%
State resetting	5.93%	4.51%	35.47%	36.92%
Data loading&storing	45.55%	48.09%	23.10%	14.30%
Others	11.89%	7.76%	0.40%	0.69%

gesture recognition data sets by collecting EMG signals and spike signals captured by DVS and used them to evaluate Loihi, ODIN, and MorphIC. This is the first time that LRW is used for neuromorphic processor evaluation.

1) Execution Time Breakdown: To find the operators that need to be defined as instructions, we conduct observation on execution time breakdown on applications. As shown in Table I, we analyze the average percentage of different operators' execution time in the total execution time of the three datasets running on the above three SNNs on the host with I7-10700 K CPU using VTune profiling software. We conclude that the current accumulation operation for calculating the neuron current, a series of operations for calculating the neuron model, and the data loading and storing operations occupy the main part of the program execution time, which are 11.88%−29.47%, 13.31%−73.13%, and 14.30%−48.09% respectively. Thus, these operations need to be defined in the neuromorphic ISA to be supported and accelerated. Note that the "others" part in the table includes general-computing operations and execution processes related to program execution branches and jumps, which also shows from one side that control processes and other general computations other than SNN computing occupy an indispensable position in the SNN program.

2) Dynamic Sparsity: The input spike train will change dynamically in each time step in the execution process of the neuromorphic application and dynamic sparsity exists in the spike train. Thus, to guide suitable parallelism granularity of the instructions defined in the ISA, we analyze the sparsity in the input spike train of spiking VGG & MobileNet and spike train generated by hidden-layer neurons in LSM during each time step of running datasets, i.e. the number of valid spikes generated by fired neurons.

As shown in Fig. 4(a) and (b), the granularity of 32 can cover about 90% and 50% conditions in LSM with 343 hidden-layer neurons and 1000 neurons respectively and granularity of 128 can cover about 90% conditions in LSM with hidden-layer 1000 neurons. As for the spiking VGG and MobileNet, since they have large-scale input layers and hidden layers, the number of fired input neurons shown in Fig. 4(c) and (d) are much bigger than the LSM. Thus a load instruction with a much bigger loading data width such as 512 bits is needed to accelerate the data loading process.

Thus, the numbers of spikes and weights loaded by defined instruction in the neuromorphic ISA need two typical granularity, 32 and 128 to cover the most requirements in LSM. For a better understanding of the relationship between granularity and sparsity, we give an example. As shown in Fig. 4, the number of activated neurons is less than or equal to 32 in 90% of the time

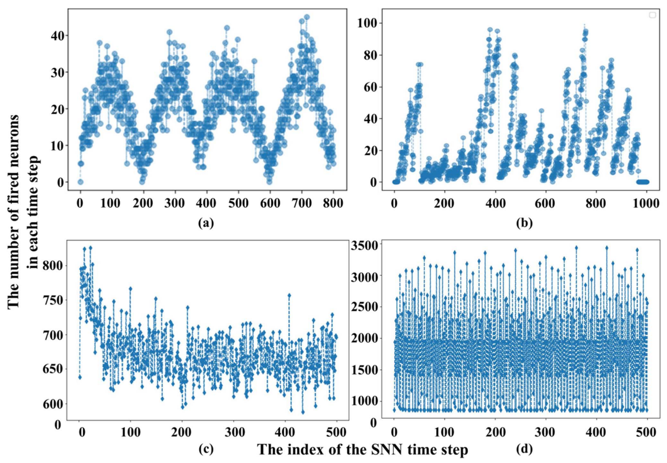


Fig. 4. The number of fired neurons in (a) 343 hidden-layer neurons in LSM when running DVS128 Gesture dataset (b) 1000 hidden-layer neurons in LSM when running LRW lip reading dataset (c) 32*32*3 input neurons in spiking VGG-11 when running CIFAR-10 dataset (d) 32*32*32 neurons in the first layer of spiking MobileNet V1 when running CIFAR-10 dataset.

steps. Therefore, for spike transmission and neuron computation, the granularity of 32 can cover the computation requirements in 90% of the time in the inference of an input sample.

C. ISA Specification

Based on the profiling results of typical SNN algorithms and datasets in the last section, we find the most-used operators that need to be supported in the ISA. In this section, we introduce the specification of RVNE and the example programs using instructions in RVNE.

We have designed a total of 8 types of instructions, which are wide-vector loading (including weights, spikes, and parameters), current accumulation, states updating, spike moving, state storing, and multiplication and accumulation (MAC) for classification. Note that opcodes of all types of RVNE instructions use the custom fields reserved in RISC-V specification (Volume I: User-Level ISA Document Version 2.2).

The computational instructions of RVNE include instructions with different granularity and dataflows. Thus various degrees of sparsity in SNN can be exploited in different ways. Besides, to speed up the loading and storing speed for SNN and balance the computing and irregular memory accessing caused by unevenly distributed activated neurons in each time step shown in Fig. 4, we design the wide-vector load and store instruction to fetch or save data from and to memory.

1) Extended Vector Registers: To store the operands required for neuromorphic application execution, we define the following types of extended vector registers.

Weight vector register (WVR): To store fan-in synaptic weights of neurons, we define WVR. It can be used as the target register of weight-loading instructions. In weight-loading instructions of different granularity, 1 WVR or 1 group of WVRs can be loaded with weights from the memory. Current accumulation instructions can read the operands in WVR for computation.

Spike vector register (SVR): To store the input spike trains, we define SVR. It can be used as the target register for input spike loading instructions. In different granularity of spike loading instructions, 1 SVR or 1 group of SVR can be loaded with input spikes from the memory. The current accumulation instruction can read the contents of the SVR as the operands.

Spike output register (SOR): To store the output spikes generated by hidden-layer neurons, we define SOR. It can be used as the target register of states updating instructions to receive and store the output spikes generated by neurons during the neuron states updating process. Spike moving instruction can read the content in SOR and transfer it to the SVR to realize the transfer of spikes from layer to layer or from time step to time step.

Neuron state register (NSR): To store the states including the number of times a neuron fires in a period, neuron current, neuron voltage (membrane potential), and refractory period of a neuron, we define NSR. It can be used as a target register for state-updating instructions to receive and store the updated states of neurons after the neuron model computation is performed. The states updating and MAC instructions can read its value as operands to perform neuron model computation and classification. The state-storing instruction can read its value as the contents to write back to memory.

2) Extended Neuromorphic Instructions: Weight Loading To load multiple fan-in weights of a neuron into WVR, we define weight loading instruction whose format is shown in Table II. Based on a given base and shift memory addresses stored in $rs1$ indexed register and imm field, weight loading instruction controls to load multiple fan-in weights of a neuron into WVR indexed by content stored in rd indexed register. Note that the reason we use indirect register indexing is that it can reuse static assembly code to achieve assembly code loops and a significant reduction in static code size. A hint field is also reserved to indicate whether the memory needs to prefetch data continuously or not. This class of instruction has the following three granularities.

lw.wv/lh.wv/la.wv: Load 8/32/128 fan-in weights to 1/4/all WVR(s) indexed by rd.

Spike Loading: To load multiple input spike trains during time steps we define spike loading instructions whose format is shown in Table II. Based on given base and shift memory addresses stored in $rs1$ indexed register and imm field, spike loading instruction controls to load multiple input spike trains into SVR indexed by content stored in rd indexed register. A hint field is also reserved for the same purpose mentioned above. This class of instruction has the following three granularities.

lw.sv/lh.sv/la.sv: Load 32/128/512 input spikes (1-bit each) into 1/4/16 SVR(s) identified by rd.

Parameter Loading: To load neuron parameters, such as refractory period, threshold voltage, neuron types (excitatory or inhibitory), etc, we define parameter loading instruction whose format is shown in Table II. Based on base and shift memory addresses stored in $rs1$ and $rs2$ indexed registers, neuron parameters are loaded in registers indexed by content stored in the rd indexed register. This class of instruction has the following three instructions of different types.

TABLE II
THE FORMATS OF ALL NEUROMORPHIC INSTRUCTIONS IN RVNE

synaptic weight loading instructions					
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**		**opcode[6:0]**
offset	base	lw.wv	WVR_dst		CUSTOM
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**hint[0]**	**rd[3:0]**	**opcode[6:0]**
offset	base	lh.wv	hint	WVR_dst	CUSTOM
offset	base	la.wv	hint	WVR_dst	CUSTOM
spike vector loading instructions					
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**hint[0]**	**rd[3:0]**	**opcode[6:0]**
offset	base	lw.sv	SVR_dst		CUSTOM
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**hint[0]**	**rd[3:0]**	**opcode[6:0]**
offset	base	lh.sv	hint	SVR_dst	CUSTOM
offset	base	la.sv	hint	SVR_dst	CUSTOM
neuron states and parameters loading instructions					
funct7[6:0]	**rs2[4:0]**	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
000 0000	offset	base	lw.rp	00000	CUSTOM
000 0001	offset	base	lw.vt	00000	CUSTOM
000 0010	offset	base	lw.nt	dst	CUSTOM
neuron current computing instructions					
funct7[6:0]	**rs2[4:0]**	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
111 0000	SVR_src	WVR_src	convh	NSR_dst	CUSTOM
111 0001	SVR_src	WVR_src	conva	NSR_dst	CUSTOM
111 0010	SVR_src	WVR_src	convmh	NSR_dst	CUSTOM
111 0011	SVR_src	WVR_src	convma	NSR_dst	CUSTOM
111 0100	SVR_src	WVR_src	doth	NSR_dst	CUSTOM
111 0101	SVR_src	WVR_src	dota	NSR_dst	CUSTOM
neuron states updating instructions					
funct7[6:0]	——	——	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
1110100	00000	00000	upds	NSR_src/dst	CUSTOM
1110101	00000	00000	updg	NSR_src/dst	CUSTOM
1110110	00000	00000	upda	NSR_src/dst	CUSTOM
spike vector moving instructions					
funct7[6:0]	——	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
111 0111	00000	SOR_src	movg	SVR_dst	CUSTOM
111 1000	00000	SOR_src	mova	SVR_dst	CUSTOM
neuron states storing instruction					
funct7[6:0]	**rs2[4:0]**	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
000 0000	NSR_src	base	sa.ns	offset	CUSTOM
classification computation instruction					
funct7[6:0]	**rs2[4:0]**	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
111 1001	NSR_src	GPR_src	mac.ns	offset	CUSTOM

a) *lw.rp:* Load the refractory period thresholds of exciting/inhibiting neurons into parameter registers (refractory period register, RPR).

b) *lw.tv:* Load the threshold voltages of exciting/inhibiting neurons into parameter registers (voltage threshold register, VTR).

c) *lw.nt:* Load a group of the types of exciting/inhibiting neurons into parameter registers (neuron type register, NTR) indexed by the rd field. Note that using 1 b can indicate whether the neuron type of a neuron is exciting or inhibiting.

Current Accumulation: The numbers of neurons used in different neuromorphic applications are various and the current accumulation of neurons can be conducted in two different ways. In lightweight tasks, only a small number of neurons need to be used. So in this case the neuron-wise way of computation is suitable to be adopted as shown in the left part of Fig. 5. Synaptic operations (SOPs) of multiple weight connections of one neuron using one instruction at a time. Only the currents of used neurons will be updated, while the other neuron units can be closed to save energy.

In a heavy neuromorphic workload, a large number of neurons need to be used. So it is necessary to adopt the synapse-wise way of computation as shown in the right part of Fig. 5. The synapse-wise way of computation calculates the synaptic operations of multiple neurons connected to the neuron which outputs a spike using one instruction at a time. Note that the synapse-wise way of computation is suitable for the AER data processing and

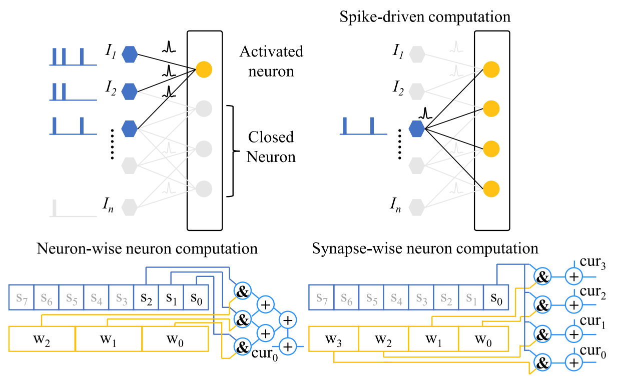


Fig. 5. Different dataflows (in the synapse-wise way and in the neuron-wise way) of neuron current computation. AER data can be processed in the synapse-wise way and the event-based feature of neuromorphic computing can thus be realized.

the event-based feature of SNN can thus be realized using this computing flow.

Thus, to support two ways of current computation to exploit the sparsity in two perspectives of SNN we define different types of instructions whose formats are shown in Table II. Their functions are to read the specified WVR and SVR content, perform the synapse-wise way or the neuron-wise way of the current accumulation computation, and update the current(s) of specified neuron(s). This class of instructions has the following six instructions with six different granularity and two ways.

a) *convh/conva:* Read 32/128 weights in a set of WVR indirectly indexed by $rs1$, 32/128 1-bit spikes in an SVR indirectly indexed by $rs2$, and the current of a neuron stored in the neuron state registers indirectly indexed by rd respectively and perform the neuron-wise synaptic operation and then the current accumulation with a parallelism of 32/128 to update the current of a specified neuron.

b) *convmh/convma:* Read 128 weights in a set of WVR indirectly indexed by $rs1$, 128 1-bit spikes in an SVR indirectly indexed by $rs2$, and the currents of 16/4 neurons stored in the neuron state registers indirectly indexed by rd respectively and perform the multiple neuron-wise synaptic operations and then the current accumulation with a parallelism of 128 to update the currents of a set of specified neurons. These two instructions are designed for developing neural networks with decreasing sizes of weight kernels (filters) such as MobileNet to increase the computation parallelism.

c) *doth/dota:* Read 32/128 weights in a set of WVR indirectly indexed by $rs1$, a 1-bit spike in the first SVR indirectly indexed by $rs2$, and a set (32/128) of neuron currents indirectly indexed by rd and perform the synapse-wise synaptic operation and addition to update the currents of 32/128 specified neurons.

States Updating: To perform operations including current leakage, voltage leakage, voltage update, threshold comparison, spike generation, state resetting, and other necessary operations according to the neuron formulas to update the states of specified neurons, we define the states updating instructions whose formats are shown in Table II. This class of instructions has the following three instructions of different granularity.

upds/updg/upda: Perform the states updating of all states including current, voltage, spike sums, and refractory period stored in the extended registers of 1/32/all neurons indirectly indexed by rd. The states updating computation follows the LIF neuron model described in equation (1) or other neuron models which can be reconfigurable in the future.

Spike Moving: To move output spikes generated by hidden-layer neurons to SVR to be used as the input spike train of the next time step or the next layer, achieving the on-chip data reusing of spike train, we define the spike moving instruction whose format is shown in Table II. This class of instructions has the following two instructions with different granularity.

movg/mova: Read 32/all spikes from an SOR indirectly indexed by $rs1$ and move them to SVR indirectly indexed by rd.

States Storing: Since the accommodated number of neurons in the extended registers is limited, to support more neurons than the extended registers can store, we define neuron state storing instruction whose format is shown in Table II to save the states of neurons. Using the given initial and shift addresses stored in $rs1$ and rd indexed registers, this instruction controls to store a set of neuron states stored in $rs2$ indirectly indexed NSR back to the memory.

sa.ns: Store the contents of a set (64) of NSR indirectly indexed by $rs2$ back to the memory.

Multiplication and Accumulation (MAC): SNN generally uses its sums of spikes (the number of times a neuron is fired in a period) as input to the fully-connected layer to perform classification calculations of patterns or targets. Therefore, to complete the end-to-end SNN computation process, we also design an instruction whose format is shown in Table II to support the MAC operation in the classification layer which can also be used for the computation in Artificial Neural Networks (ANNs). Note that the operand width of MAC instruction is different from the current accumulation designed for SNN. If MAC is used for the current accumulation, it will not only reduce the maximum parallelism but also waste the operand bit width since the common-used operand width of the spike is only 1 b.

mac.ns: Read out the contents of one general-purpose register indirectly indexed by $rs1$ as weights operands of classification layer, read out the of one NSR indirectly indexed by $rs2$ as input feature operands, read out the content of one general-purpose register indirectly indexed by rd which stores the intermediate result of the MAC and perform MAC operation.

D. Examples of SNN Implemented With Extended Instructions

As shown in Fig. 6, we give an example of the whole operation process of the LSM, a kind of recurrent SNN, written by assembly instructions defined in RVNE. It can be seen from the figure that general RISC-V instructions such as addi (immediate addition) and bne (branch not equal) and extended instructions such as conva and mova are equivalent to each other in the program, achieving flexible and fine-grained instruction-level programming of homogeneous computing.

```
1: addi a0,x0,800 #Time step index of a sample
   #Spike communication and neuron computation
    within the 343-neurons hidden_layer
2: addi a3,x0,0#Pointer of hidden-layer neurons
   addi a5,x0,343#The total number neurons
   la a2, hidden_layer_weight #The base address

3: addi a7,x0,3
   addi a4,x0,0#Pointer of spike trains

4: la.wv x0,0(a2)#Load hidden-layer weight
   conva a3,a4#Neuron current accumulation
   addi  a2,a2,1#Shift address of hidden-layer weights
   addi  a4,a4,1
   bne   a4,a7,4b#Branch to No.4 programm segment
   addi  a3,a3,1
   bne   a3,a5,3b

   #Spike communication of 4096-neurons input layer
   addi a3,x0,0#The pointer of hidden-layer neurons
   addi a5,x0,343#The total neuron of the hidden-layer
   la a1, input_spike_train#The base address
   la a2, input_to_hidden_layer_weight #Base address

5: addi a6,x0,8

6: la.sv x0,0(a1)#Load input spike train
   addi a7,x0,4
   addi a4,x0,0#The input spike train pointer

7: la.wv x0, 0(a2)#Load weights
   conva a3,a4#Neuron current accumulation
   addi a2,a2,1#Weights address increases
   addi a4,a4,1#Spike trains pointer increases
   bne a4, a7, 7b
   addi t2,x0,1
   sub a6,a6,t2
   #Increase the shift address of input spike train
   addi a1,a1,1
   bne x0, a6, 6b
   addi a3,a3,1 #Point the next neuron
   bne a3, a5, 5b
   #Computation according to the neuron model
   upda
   #Move all output spikes in SOR to SVR
   mova
   sub a0,a0,t2
   #Branch back to perform the time step
   bne x0, a0, 2b
   ecall

.data
.globl  input_to_hidden_layer_weight
.globl  hidden_layer_weight
.globl  input_spike_train
```

Fig. 6. Application of running DVS128 Gesture dataset using LSM implemented in assembly instructions defined in RVNE.

IV. NeuroRVcore

Compared to the implementations of accelerators mounted on the off-chip/on-chip bus, homogeneous computing implements function units that are tightly coupled in the instruction pipeline. This is more friendly for software programming because there is no essential difference between extended instructions and general instructions to the compilation toolchain. To demonstrate the RVNE, we propose a neuromorphic hardware implementation in the instruction pipeline to accelerate neuromorphic computing. Next, we will talk about the details.

A. Instruction Pipeline

NeuroRVcore is implemented based on an open-source RISC-V core named RI5CY [13]. As shown in Fig. 7 the instruction pipeline of NeuroRVcore contains four stages, instruction fetching (IF), instruction decoding (ID), executing (EXE) and writing back (WB), and three buffer stacks, IF/ID, ID/EXE, and EXE/WB.

The function of the instruction pipeline is to complete the function sets defined by the instructions. Its workflow is as follows: the first step is to fetch instructions from the instruction cache in the IF stage. Second, in the ID stage, according to the definition of different fields in the instruction, the indexed operands in the specified register are read out. Third, in the EXE stage, according to the instruction opcodes and function fields defined in the instruction, the specified function is performed using the operands read from the ID. If the instruction is load or store, data will be stored in memory through the load and store unit (LSU), or a loading request is sent to memory through LSU. Fourth, in the WB stage, the calculation result of the previous stage or the data loaded from memory is written back to a specific register(s).

1) IF: The IF stage contains an instruction prefetch buffer, which is responsible for fetching and caching instructions from the instruction cache, and a debugger that controls the debugging of the processor in debug mode through an external debug interface.

2) ID: The ID stage includes a decoder which is responsible for translating the register indexes and control signals stored in different fields of instructions and sending them to the register groups including general-purpose registers (GPR) and extended vector registers. It also contains multiple multiplexers that are used to choose indexes from different instruction fields. The organization of extended vector registers is introduced in Section III-C1 in detail.

3) EXE: For general-purpose computation in RV32IMC, the EXE stage contains an LSU, which is responsible for processing the request of storing and loading, an arithmetic logic unit (ALU) that performs general functions such as addition and subtraction, a multiply and division (MUL/DIV) unit that performs complex multiplication and division operations, a control state register (CSR) that is responsible for storing signals related to control, such as interrupts, exceptions, etc, multiple multiplexers that choose the source operands of computation, a controller used to judge whether to perform branch or execution in sequence.

For neuromorphic computing, the EXE stage consists of a neuron array composed of 1024 neuron units (NUs) which perform neuron computation according to the formula 1 in a multi-cycle way, a vector load store unit (VLSU) responsible for processing the requests of vector storing & loading and neuron/synapse-wise current accumulators to perform acceleration as shown in the right part of Fig. 7 discussed next in Section IV-B.

4) WB: The WB stage contains multiple multiplexers for selecting the results of the EXE stage or data read from memory to write back to specific registers and a ScratchPad Memory (SPM) that responds to storing requests and receives data transferred from LSU to store in the memory to achieve high memory bandwidth, balancing computing and irregular memory access.

B. Synapse-Wise and Neuron-Wise Current Accumulator

As discussed before, the EXE stage also consists of neuron-wise and synapse-wise current accumulators to perform current accumulation computing acceleration with multi-granularity parallelism, exploiting different granularity sparsity utilization.

The synapse-wise current accumulator is the key to efficiently processing AER data and thus approximately implements the event-driven feature based on synchronous circuit implementation. In the synapse-wise way of computation, the accumulator calculates the synaptic operations of multiple neurons using one instruction at a time. In this way, once there is an AER data, i.e. an input spike, only the synaptic weights connected to the input

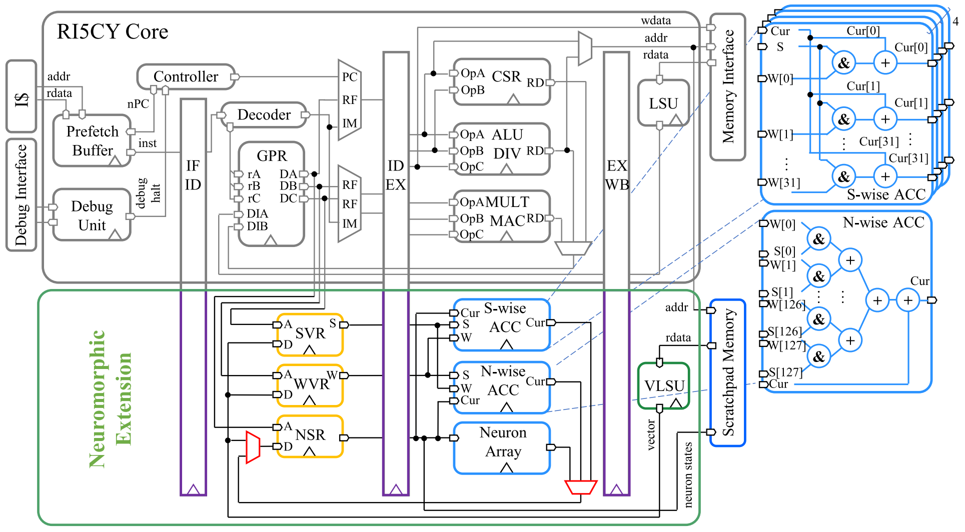


Fig. 7. Block diagram of the microarchitecture of the NeuroRVcore.

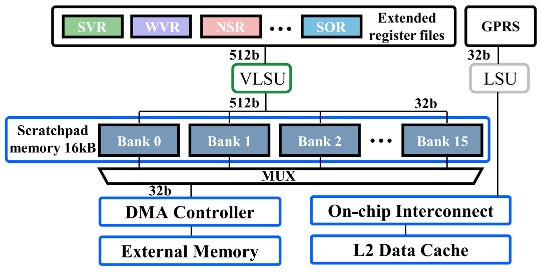


Fig. 8. Block diagram of the scratchpad memory in NeuroRVCore.

neuron that receives this spike will be taken out into the WVR. Then the synaptic operations of these synapses will be executed in parallel according to the specified parallel granularity according to the instruction. Finally, the destination neurons of these synapses will be updated. Otherwise, if there is no input AER data, the above operations will not be executed at all to achieve the event-driven low-power feature and save energy.

In the neuron-wise computing way, the accumulator calculates synaptic operations of multiple synaptic connections of one neuron using one instruction at a time. The non-zero synaptic weights of a neuron are traversed by invoking the neuron-wise computation instruction multiple times to finally obtain the current accumulation value and update the current state of a neuron. In this way, only the required number of neurons in specific applications are allowed to operate while parallel computation of synaptic connections is also ensured. Other neuron units in the neuron array can be closed to achieve low power and save energy.

C. Scratchpad Memory

The “memory wall” problem is the challenge brought about by adopting homogeneous neuromorphic architecture. To alleviate unbalance between irregular memory accessing and neuromorphic computation, as shown in Fig. 8, an extra scratchpad memory is proposed to provide high memory bandwidth. Accordingly, the wide-vector load and store instructions defined in RVNE can directly access the scratchpad memory through the wide-vector memory accessing controller to get the data.

The scratchpad memory is similar to the cache in function but is simpler since it does not need to store tags. It is arranged as a segment of storage space identified by a physical base address and a contiguous space immediately following the base address and it can fast load a segment of memory space at one time through direct memory access (DMA) when it is enabled. Frequently-used data such as synaptic weights can be accommodated in scratchpad memory to achieve high data reusing and alleviate the “memory wall” problem.

To trade off the hardware cost and memory access latency, we divide the scratchpad memory into 16 banks where each bank has a 32-bit data read port and a data write port. The size of the scratchpad memory is 16 KB. In this way, the wide-vector load and store defined in RVNE interact directly with scratchpad memory with 512-bit data bandwidth. Besides, to reduce the hardware cost, the scratchpad memory does not implement the hardware-guaranteed data coherence. To avoid the data coherence problem, when the data is moved from the external memory to the scratchpad memory, guaranteed by the software, the data cannot be accessed by the LSU unit within the main pipeline.

In particular, the workflow of scratchpad memory is to first receive data loaded from external memory controlled by DMA before the computation begins. After that, the extended load instruction loads data into the extended registers from scratchpad memory which serves as the local memory. When the computation is completed, the extended store instruction stores the result back to scratchpad memory and waits for the DMA operation or continues the subsequent calculation process.

V. Experiment

A. Implementation

We implement all instructions defined in RVNE using QEMU [46], a RISC-V instruction simulator, and build an assembler for RVNE to compile its assembly instructions into

TABLE III
THE COMPARISON OF CODE DENSITY (ASSEMBLY CODE LENGTH) BETWEEN RV32IMC AND RVNE

	RV32IMC		RVNE	
	Static	Dynamic	Static	Dynamic
LSM/DVS128	2419	1.2×10^{10}	612	5.1×10^{6}
LSM/LRW	2529	9.2×10^{12}	986	4.2×10^{9}
SVGG/CIFAR-10	2426	4.6×10^{14}	564	3.9×10^{11}
SMOB/CIFAR-10	3518	9.6×10^{10}	668	7.6×10^{8}

TABLE IV
THE NUMBER OF RV32IMC INSTRUCTIONS EQUIVALENT TO RVNE INSTRUCTIONS

RVNE	Ins. Num.	RVNE	Ins. Num.	RVNE	Ins. Num.
lw.wv	1	lw.sv	1	mova	32
lh.wv	4	lh.sv	4	sa.ns	16
la.wv	16	la.sv	16	convh	56
lw.rp	1	movg	1	conva	56
lw.vt	1	upds	237	doth	55
lw.nt	1	updg	64	dota	55
mac.ns	2	upda	60	-	-

binary executable files. For the prototype, we implement the micro-architecture of NeuroRVcore by adding all instructions of RVNE in the decoder and adding functional units in the execution stage of the instruction pipeline using RTL-level Verilog based on the open-source RISC-V Core, RI5CY [13]. The newly added components do not affect the core operating frequency in 1 GHz for target applications. Even if they are in the critical path, they can be pipelined to achieve a higher frequency. For hardware validation, the micro-architecture verification environment is UVM [47]. The synthesis and area evaluation of the prototype is under 28 nm technology.

B. Experiment Setup

The SNN simulator software frameworks we use to generate and simulate SNNs are Brian2 [48] and SpikingJelly [49]. The datasets we use in the SNN profiling are the DVS128 hand gesture dataset [22], LRW lipreading dataset [23], and CIFAR-10 dataset [24] which are converted into the formation of spike trains according to Poisson distribution.

The DVS128 gesture dataset consists of 11 hand gestures from 29 subjects under 3 illumination conditions, captured by a DVS camera. Each sample in it is a spatio-temporal spike train with a duration of about 800 ms and a frame resolution of 32 × 32 pixels. LRW dataset is an English word-level lipreading dataset. It contains 500 English words and over 500, 000 utterances. The content of it is mainly short video clips of BBC news talks (1.16 seconds each). The CIFAR-10 dataset consists of 60000 32 × 32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The LSM used for the DVS128 hand Gesture dataset inference consists of 343 neurons and 4096 input neurons. The LSM used for the LRW inference consists of 1000 neurons and 4096 input neurons. The SCNN used for CIFAR-10 inference is the spiking version of VGG-11 and MobileNet V1.

For hardware, we use Verilator to perform the functional simulation. The power evaluation is performed by inputting the waveform and the hardware library to EDA tools. For related work comparisons, we use the same SNN and datasets.

C. Experiment Result

1) Code Density: To prove the effectiveness of RVNE, we use instructions of RV32IMC and RVNE to write the above four neuromorphic applications and get the static code density comparison as shown in Table III. We also give the numbers of equivalent assembly instructions of RV32IMC to RVNE as shown in Table IV.

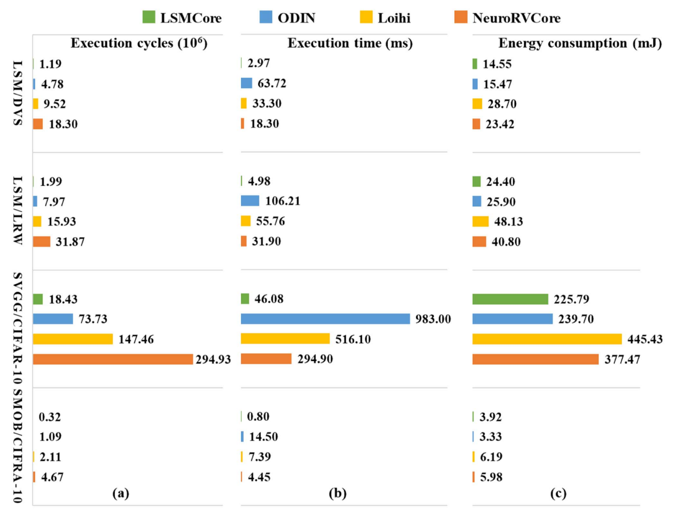


Fig. 9. The (a) execution cycles, (b) execution time, and (c) energy consumption comparison between NeuroRVcore and neuromorphic processors such as LSMCore, ODIN, and Loihi on four neuromorphic applications (running DVS128 Gesture and LRW lip reading datasets on LSM and running CIFAR-10 datasets on spiking VGG and MobileNet).

Experimental results show that the static code density of the above four neuromorphic applications implemented using RVNE can be reduced by $2.8\times - 5.3\times$ compared with the RV32IMC basic instructions set.

2) Energy and Execution Time: We evaluate and calculate the execution cycles, execution time, and energy consumption of NeuroRVcore and other neuromorphic processors such as LSMCore, ODIN, and Loihi 1 as shown in Fig. 9. Since LSM only has one hidden layer while converted networks such as spiking VGG and spiking MobileNet have multiple hidden layers, we show one of the hidden layers in both spiking VGG (the sixth layer) and spiking MobileNet (the third layer). And batch normalization computation is not included in the experiment results because LSMCore and ODIN can not support it. Note that only the performance results of the computation process are shown in the figure and the communication between CPU and co-processors is not included.

Among the neuromorphic processors we use to compare, ODIN is a 0.086 mm^2 64k-synapse 256-neuron online-learning digital neuromorphic processor in 28 nm FDSOI CMOS. LSMCore is an 18.49 mm^2 neuromorphic processor in 40 nm CMOS for LSM inference and online-training acceleration and supports 256 input neurons, 1024 liquid neurons, and 1.31 M synapses.

TABLE V
THE COMPARISON BETWEEN NEURORVCORE AND THE STATE-OF-THE-ART NEUROMORPHIC PROCESSORS

	TrueNorth [5]	Loihi [6]	DYNAPs [2]	MorphIC [52]	shenjing [53]	LSMCore [54]	**This work**
Technology(nm)	28	14	180	65	28	40	**28**
Implementation	Async.	Async.	Async.	Sync.	Sync.	Sync.	**Sync.**
Frequency(Hz)	N.A.	285.7M	400M	210M	243M	400M	**1G**
Power(mW)	63-300	N.A.	N.A.	N.A.	1.35-887.8	4900	**1280**
Performance(SOPs)	58G	35.7G	400G	420M	121.5G	400G	**128G**
Area(mm^2)	430	60	43.79	2.86	0.49	18.49	**2.52**
Neurons	1M	127K	1K	2K	2.5K	1K	**1K**
Synapses	512M	128M	64K	2.1M	N.A.	1.31M	**32K**
Energy efficiency(SOPs/W)	193G-920G	N.A.	N.A.	N.A.	137G	81.6G	**100G**
ISA Support	×	x86	×	×	×	×	**RISC-V**

Loihi is a 60 mm^2 online-learning neuromorphic processor fabricated in Intel's 14 nm process, which supports 128 K neurons and 128 M synapses.

The execution cycles of related work are obtained by dividing the number of total SOPs by the computational performance. Since asynchronous circuits do not have the concept of the clock cycle, for comparison, we take the time of an SOP mentioned in published papers, such as 3.5 ns in Loihi, as a clock cycle equivalent to synchronous circuits. The execution time of related work is calculated using the product of execution cycles and their frequencies. The energy consumption of the related work is calculated by using the product of their power and their theoretical execution time or the product of their energy consumption of a single SOP given in their papers and the number of theoretical operations. The power comparison of them is shown in Table V.

As can be seen from Fig. 9(a), the execution cycle of NeuroRVcore is longer than that of LSMCore, ODIN, and Loihi 1 because NeuroRVcore is limited by storage bandwidth and needs to use load/store instructions to get data. But due to the high frequency of NeuroRVcore, it needs less execution time than ODIN and Loihi 1 whose frequency is 75 MHz and equivalent 285.7 MHz as shown in Fig. 9(b). Thus NeuroRVcore achieves a 1.66 × −1.82 × and 3.26 × −3.48 × speed up when compared with Loihi 1 and ODIN respectively. The reason is that ODIN and Loihi 1 use the method of time-division multiplexing of neurons and their frequencies are much smaller than NeuroRVcore. As shown in Fig. 9(c), regarding the energy consumption, NeuroRVcore consumes 3.4%−22.5% less energy than Loihi 1 which consumes 23.6 pJ per SOP.

The execution time and energy consumption of LSMcore are better than those of NeuroRVcore because the experimental results only consider the computation process and LSMCore contains more neuron units and larger memory bandwidth compared to NeuroRVCore which incurs a larger area and power consumption as shown in Table V in detail. Compared to LSMCore without ISA, NeuroRVCore outperforms LSMCore in programmability because it provides a unified programming interface for decoupling neuromorphic software and hardware and thus provides sufficient headroom for upper-level compilation optimization by enabling finer-grain instruction-level parallelism. Besides, neuromorphic processors such as LSMCore and ODIN can not support batch normalization computation included in state-of-the-art neural networks such MobileNet. Thus, this kind of computation needs to be off-loaded to the main CPU, leading to frequent data exchange and significant data mobility overhead. If we consider this factor, NeuroRVCore could gain better performance in the experimental results.

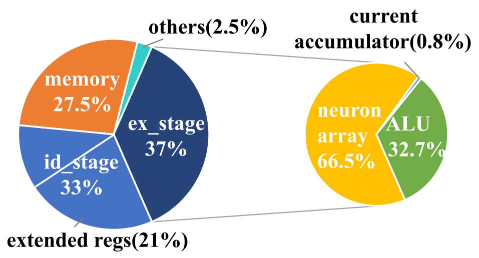


Fig. 10. The area breakdown of NeuroRVcore.

What's more, considering the influence of software, the deeper software stack and hardware driver invoking required for heterogeneous architectures such as Loihi and ODIN leads to higher latency, thus showing less energy efficiency when facing the same applications compared with the homogeneous architectures.

3) Area Breakdown: As shown in Fig. 10, we present the area breakdown of NeuroRVcore. It can be seen that the execution stage, extended memory, and registers take up 73% of the area. In the execution stage, the extended modules, i.e. the neuron array and the current accumulator, also occupy more than half of the area. The area of RI5CY under 28 nm technology node is 0.017 mm^2 and accounts for 0.67% of NeuroRVCore. The energy consumption of these two parts is 1.3% and 98.7% of the whole, respectively.

4) Implementation Efficiency Analysis: As shown in Fig. 11 below, we studied the relationship between peak performance and the parallel granularity of synaptic computing units and the efficiency using different configurations. We use area efficiency to represent the efficiency index, which can represent energy efficiency to a certain extent.

We studied five architectures with different numbers of parallel granularity of the synaptic computing units. As shown in Fig. 11, point named 128 represents an architecture configuration whose parallel granularity of the synaptic computing unit is 128, while other architecture parameters of the point include 512 b synaptic weight vector extended register, 1024 b spike vector extended register, 1024 neuron units, and associated 1024 neuron state extended register. The granularity of each parameter in the other four architecture configurations decreases by one time in turn, as that of synaptic computing parallelism does. The frequencies of all architectures are 1 GHz by default.

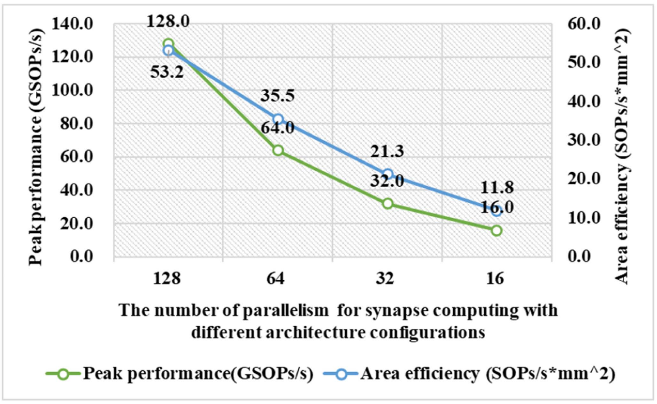


Fig. 11. The peak performance and area efficiency of architectures with different synapse computing parallelism and corresponding changed architectural configurations.

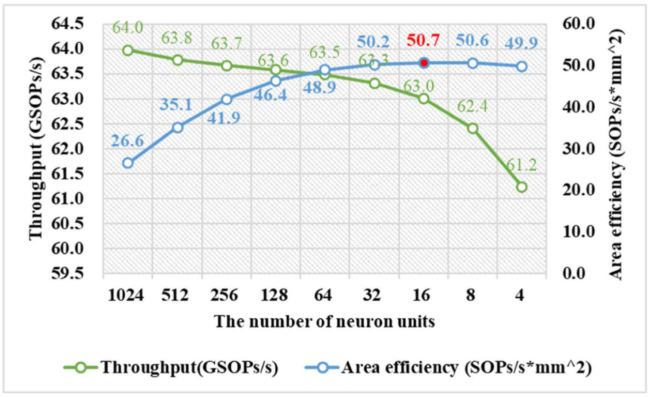


Fig. 12. The throughput and area efficiency of architectures with the different number of neuron units and associated neuron states extended registers. The red dot represents the inflection point of area efficiency.

The information we can get from Fig. 11 is that the peak performance is directly determined by the computing parallel granularity of synaptic units, so the relationship between them is linear. Second, because the area cost of other logic implementations is increasingly difficult to amortize to the linear-reducing peak performance, the area efficiency continues to decrease. On one hand, although architectures with the parallel granularity of less than 128 can achieve higher working frequency and thus achieve higher peak performance, the increase in frequency is not always linear. Thus it is difficult to achieve higher frequencies only by simply reducing the number of multilevel addition trees in synaptic computing units. On the other hand, when we want to achieve higher peak performance by increasing the parallel granularity of synaptic computing units, increasing the number of addition trees will become the critical path to affect the upper limit of frequency. Although we can implement this critical path through multi-level pipelining, this will also increase the complexity of the implementation, and multi-level pipelining essentially reduces the completion frequency of the synaptic computing operation.

As shown in Fig. 12, we also studied the relationship between the number of neuron computing units and the system throughput & area efficiency, to find the number of neuron units that can achieve the best performance/area compromise. Since the relationship between the number of neuron computing units and performance cannot be characterized by peak SOPs, we choose the number of synaptic operations completed in a time step, namely throughput, to characterize the performance.

We have studied 9 architectures with different configurations. The only two differences between them are the number of neuron computing units and the number of associated neuron state registers. The network we use for this quantitative analysis is an LSM with 1024 hidden layer neurons and 1024 input neurons. Note that if the number of neurons in the hidden layer exceeds the number of neuron computing units and associated state registers in the architecture, the neuron state variables will be loaded from the memory in each time step, calculated, and written back to memory to be stored. This process will incur performance overhead.

From Fig. 12, we can get the following information. First, from the perspective of performance, reducing the number of neuron computing units will increase the time of neuron computing and consume extra time on neuron state loading and storing. For the architecture with 1024 neuron units, all neuron state variables can stay in the extended register in all time steps, so there is no need to load and store them from and to the memory at each time step. From Fig. 12, we can also see that the throughput of different architectures has gradually decreased, but the range is not large because the proportion of neuron computing and state loading and storing in one time step execution time is relatively low.

From the perspective of the area, reducing the number of neurons can effectively reduce the area cost. Therefore, as the number of neurons continues to decrease, the area efficiency continues to improve. It is worth noting that when the number of neurons is 16, the area efficiency reaches the inflection point. This is due to the significant reduction in the number of neuron units, the proportion of the time of neuron computing and state loading/storing in the total execution time has increased significantly, and the efficiency improvement brought by the reduction in the area has appeared marginal diminishing effect. Therefore, we can conclude that for the target network described above, implementing 32 or 16 hardware neuron computing units and corresponding associated state registers can achieve the best efficiency and finally reduce the total area.

5) Comparison: As shown in Table V, we synthesize the implementation of NeuroRVcore and give the power, performance, area (PPA), etc. comparison between NeuroRVcore and the state-of-the-art neuromorphic processors.

When NeuroRVcore supports the same magnitude of neurons and synapses as other neuromorphic processors such as LSMCore [52] does, the area overhead and power consumption of NeuroRVcore are smaller than LSMCore. The most outstanding advantage of NeuroRVcore is that it has the support of open-source RISC-V ISA and thus it is easy to program and use. Although Loihi has the support of the x86 ISA, it must be used in a difficult heterogeneous programming way. We also compare NeuroRVCore with a tightly-coupled vector processor with SIMD instructions such as MPIC [21] which outperforms ARM Cortex M7 in mixed-precision DNN edge inference. Since

TABLE VI
THE COMPARISON BETWEEN NEURORVCORE AND ODIN UNDER THE SAME ARCHITECTURE CONFIGURATION

	ODIN [56]	NeuroRVCore
Technology (nm)	28	28
Implementation	Synchronous	Synchronous
Frequency (Hz)	75M	75M
Power (mW)	122	101
Performance (SOPs)	9.4G	9.4G
Area (mm2)	0.09	0.16
Neurons	256	1K
Synapse	64K	32K
Energy efficiency (SOPs/W)	77G	93.1G
Programmability	No	**Yes**
General computation	Not support	**RV32IMC**

the peak performance of MPIC is 6.5 MAC/cycle and it does not fully exploit the SNN feature, our design achieves 19.7 $\times$ speed up in peak performance.

For homogeneous system comparison, we compare NeuroRVCore with SpiNNaker [3] and the similarities and differences between them are as follows. As the only homogeneous system in related work, first of all, SpiNNaker is a large multi-core system for large-scale brain simulation applications. NeuroRVCore is currently a single-core homogeneous system with tightly-coupled neuromorphic acceleration execution units, which is for SNN applications in edge computing scenarios. Second, the Spinnaker mainly focuses on how to use multiple general CPUs to build neuromorphic computing systems, while we mainly focus on how to extend the structure of a single general CPU to build neuromorphic processors. Therefore, in terms of specific indicators that the two architectures focus on (such as spinnaker focuses on inter-core communication delay, while we focus more on synaptic operation performance), the quantitative comparison between them is difficult to be effective and meaningful. Overall, SpiNNaker provides a good example for us to expand into a large neuromorphic system in the future.

As shown in Table VI, for the comparison with the state-of-the-art neuromorphic processor under the same condition (technology node and frequency), we compare NeuroRVCore with ODIN [53] to demonstrate the pros and cons of our work. The new version of NeuroRVCore also reduces the neuron computing units and associated extended neuron state registers to 32 due to the conclusion of Section V-C-4, while maintaining the supported number of neurons by putting other neuron states in SPM. The decrease in the area of new NeuroRVCore comes from the decrease of neuron computing units, neuron states registers and pipeline stack registers related to neuron computing. The reduction of static power consumption comes from the reduction of leakage current caused by area reduction. The reduction of dynamic power consumption comes from the reduction of functional components and the working frequency. From Table VI, we can conclude that the energy efficiency of NeuroRVCore with 75 MHz and 32 neuron units is more than ODIN and NeuroRVCore has better programmability and supports general computations defined in RV32IMC, enabling it to handle the edge intelligent computing tasks independently without the help of host CPU.

VI. DISCUSSION

In this section, we discuss the extensibility of our ISA, the support from our developing compiler to our ISA and micro-architecture implementation, the asynchronous implementation possibility and challenges, and at last, the optimization opportunities of our work in the future.

A. ISA Extensibility

Since neuromorphic computing is still in the developmental stage, RVNE is only an application-specific ISA at present but leaves sufficient room for extension. It is worth noting that RVNE has taken into account the expressiveness of different SNN structures (convolution, fully connected, recurrent and batch normalization), and uses the operand bit width widely used by much neuromorphic hardware as much as possible, i.e. 4-bits (used in [52], [53], [54], [55], [56]), to achieve SNN coverage capability. Due to the scalability of RISC-V and the reserved fields in our ISA, in the future, operands of various widths and instructions of various functions can also be implemented as needed, not limited to 4 bits and the implemented functions in this work.

B. Support From Compiler for SPM

In this work, we include specialized scratchpad memory which is exposed to the compiler, making the application support complex. However, for embedded applications, this software-managed data movement between external memory and SPM ensures good time predictability, which is crucial for embedded systems with real-time requirements [57].

Second, at present, we implement the intrinsic function by combining assembly instructions and then use the intrinsic function to write C macros to handle the control and management of SPM. This implementation uses the intrinsic mechanism provided by compilers such as GCC to define new intrinsic functions without modifying the logic of the compiler itself, quickly realizing compiler support for instruction sets and microarchitectures. Similar methods are also used in other embedded processors [57]. Therefore, our method still has the advantage of ease and flexibility in SNN programming when compared with heterogeneous neuromorphic processors without instruction sets even though it includes scratchpad memory which makes application support complex.

Moreover, in the future, we will improve our compiler by introducing compiler optimization methods such as “memory coloring” [58] to support the automatic management of SPM and program optimization. In other words, automatic management of SPM supported by a compiler can further prompt the ease of programming flexibility in SNN programming.

At last, we emphasize that the major difference between our architecture and heterogeneous accelerators with SPM is the programming pattern. In other words, our architecture uses homogeneous programming to control, while heterogeneous architecture uses heterogeneous programming to control. This major difference leads to the difference at the compilation level, i.e. that the heterogeneous accelerator architectures must use a more complex heterogeneous architecture compiler like OpenCL, while our architecture only needs to add the intrinsic functions on the original compiler to achieve fast support for microarchitecture and instruction set.

C. Asynchronous Implementation

Based on the existing design, we can design and implement an asynchronous version of NeuroRVCore with the same function because we also have some developing foundation in asynchronous circuits designs [59], [60]. However, there are the following challenges which hinder our attempt.

At the RTL design level, asynchronous handshake control signals replace all clocks to drive data flow. The circuit used to implement asynchronous handshake inevitably brings more additional area overhead than synchronous circuits. At the simulation level, because there is no uniform clock, the delay of the asynchronous handshake control signal between modules must accurately match the delay through the module to ensure the correctness of the function. Due to the diversity of module types, this delay matching will bring much handwork and great challenges to functional simulation. At the implementation level, the lack of mature commercial EDA tools is the biggest challenge for the implementation of asynchronous circuits. If ones want to implement chips based on asynchronous circuits, the team needs at least one back-end group. Therefore, at present, only large companies such as IBM and Intel and universities with profound experience such as ETH can adopt asynchronous circuit technology to implement neuromorphic processors well. At the manufacturing level, due to the larger area of chips, the yield will inevitably decline.

D. Future Opportunities

Our compiler is being developed synchronously to provide more flexibility for programming our processor. In the future, we will optimize the instruction set and microarchitecture according to the requirements of the compiler including TVM scheduling and auto-tuning optimization for SNN and LLVM intermediate representation (IR) for our neuromorphic processor, to improve the application performance and demonstrate more different neuromorphic applications such as object recognition and tracking.

In addition, we consider exploring the development of massive systems with multiple cores based on the network on chip for larger chips like SpiNNaker in wilder scenarios such as data centers or cloud since we have done a lot of work in NoC of neuromorphic processors such as [61], [62], [63], [64], [65]. In detail, by implementing additional extension instructions, we can achieve processes such as inter-core spike packet generation, and spike packets storing back and loading from the memory. With the help of the extended DMA module, data transmission of inter-core spike packets between memory in the core and routers of NoC can be achieved.

In this multi-core massive system, cache coherence and communication overhead are the directions that need to be considered and focused on. Aiming at the key factor, i.e. communication, we will carry out the software and hardware co-optimization design based on the previous work on the dead-lock free network-on-chip which supports efficient multicasting routing [64] and small world link [63] for neuromorphic processor and automatic SNN mapping and communication-optimizing software framework for multi-core neuromorphic processors [61], [65], so as to achieve the goal of balancing the load of the inter-core communication link, reducing the spike message communication of the system, and ensuring the real-time performance of the neuromorphic system as we did before in our previous work.

At last, after validating the goodness of the proposal in the edge computing scenario, we wish to move to a more specific ISA (extension) which can be accepted by the RISC-V community and foundation.

VII. Related Work

A. Custom ISA for Neural Networks

These years, the industry and academia have proposed many custom instruction sets for neural network acceleration. In 2016, Liu et al. [17] proposed a new domain-customized ISA, Cambricon. Cambricon is an instruction set that integrates scalar, vector, matrix, logic, data transmission, and control instructions. The evaluation of Cambricon showed that it had strong descriptive ability across a wide range of neural networks and provided a higher code density than general-purpose ISAs. In 2019, Chen et al. [18] proposed iFPNA, a reconfigurable deep learning processor that used custom ISA for the neural network. The design was motivated by a trade-off between efficiency and flexibility. Valencia et al. [19] in 2020 proposed a custom ISA for embedded programmable processors that efficiently execute ANN. The processor architecture was extensible and supported any number of ANN layers and neurons in a single layer. In 2021, Bytyn et al. [20] presented ConvAix, an Application Specific Instruction Processor based on the implementation of the domain-custom ISA. It could efficiently execute CNN while maintaining great flexibility. Hashmi et al. [66] proposes a virtual ISA named NISA to form a level of abstraction between the learning algorithm and execution substrates such as CPUs, GPGPUs, and accelerators to allow the independent development of both. The motivation and contribution are different from our work which proposes neuromorphic ISA and corresponding hardware design to enable homogeneous computing. In other words, our work proposes a novel execution substrate to which the user can choose to translate the virtual ISA, i.e., NISA. In summary, there is no assembly-level ISA towards homogeneous computing customized for SNN algorithms so far.

B. General Instruction Set Extension for Neural Networks

In addition to using custom ISA to support and accelerate the neural network, many recent works propose the method of making extensions based on general ISAs like RISC-V. Gautschi et al. [13] 2017 proposed an open-source RISC-V processor core and introduced an instruction extension. The single instruction multiple data (SIMD) extensions and built-in L0 storage reduced shared memory access by 8 $\times$ and the competition by 3.2 $\times$. Ottavi et al. [21] in 2020 extended a state-based SIMD instruction extension based on the RISC-V to solve the problem that ISA does not support fine-grained asymmetric quantization of neural networks. The results showed that its performance and energy efficiency of it was improved by $1.1\times -4.9\times$ compared with the software-based implementation. Compared to the Cortex-M4 and M7, it delivered $3.6\times -11.7\times$ better performance and $41\times -155\times$ more energy efficiency. To conclude, there is no instruction set extended on the general ISA proposed for SNN algorithms so far.

VIII. Conclusion

Compared to the widely-used heterogeneous computing, this paper makes the first attempt to rethink the homogeneous architecture-level solution for neuromorphic computing. To fully exploit the sparsity features of neuromorphic applications, a neuromorphic instruction set named "RVNE" is proposed. Based on an open-source RISC-V core, we implement a homogeneous neuromorphic prototype. The evaluation results show that compared with the general purpose ISAs, RVNE reduces the code density. Compared with the state-of-the-art neuromorphic processors, our design brings energy consumption while providing fine-grained and flexible homogeneous programming.

Acknowledgment

Thanks Yi Wei, Yu Deng and Xiaokun Luan for their efforts in this work.

References

[1] A. Amir et al., "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7388–7397.
[2] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.
[3] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.
[4] B. Varkey et al., "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.
[5] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
[6] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.
[7] T. DeWolf, "Nengo and low-power AI hardware for robust, embedded neurorobotics," *Front. Neurorobot.*, vol. 14, 2020, Art. no. 568359.
[8] C. Yakopcic et al., "High speed cognitive domain ontologies for asset allocation using loihi spiking neurons," in *Proc. Int. Joint Conf. Neural Netw.*, 2019, pp. 1–8.
[9] V. N. Thakor, "Translating the brain-machine interface," *Sci. Transl. Med.*, vol. 5, 2013, Art. no. 210ps17.
[10] J. William et al., "IBM's POWER10 processor," *IEEE Micro*, vol. 41, no. 2, pp. 7–14, Mar./Apr. 2021.
[11] M. Weidmann, "Introducing the scalable matrix extension for the Armv9-A architecture," Arm Community, Tech. Rep., Jul. 14, 2021. [Online]. Available: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/scalable-matrix-extension-armv9-a-architecture#:~:text=Following%20the%20Vision%20Day%20announcement%20of%20Armv9-A%2C%20Arm, improvements%20to%20provide%20increasing%20support%20for%20matrix%20operations
[12] Tesla, "Tesla details dojo supercomputer, reveals dojo D1 chip and training tile module," 2021. [Online]. Available: https://www.datacenterdynamics.com/en/news/tesla-details-dojo-supercomputer-reveals-dojo-d1-chip-and-training-tile-module/
[13] M. Gautschi et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.
[14] A. Viale, A. Marchisio, M. Martina, G. Masera, and M. Shafique, "CarSNN: An efficient spiking neural network for event-based autonomous cars on the loihi neuromorphic research processor," in *Proc. Int. Joint Conf. Neural Netw.*, 2021, pp. 1–10.
[15] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I homogeneous synaptic input," *Biol. Cybern.*, vol. 95, no. 1, pp. 1–19, 2006.
[16] A. M. Andrew, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, vol. 32. Bingley, U.K.: Emerald Group Publishing Limited, 2003.
[17] S. Liu et al., "Cambricon: An instruction set architecture for neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architecture*, 2016, pp. 393–405.
[18] C. Chen, X. Liu, H. Peng, H. Ding, and C.-J. Richard Shi, "iFPNA: A flexible and efficient deep learning processor in 28-nm cmos using a domain-specific instruction set and reconfigurable fabric," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 346–357, Jun. 2019.
[19] D. Valencia, S. F. Fard, and A. Alimohammad, "An artificial neural network processor with a custom instruction set architecture for embedded applications," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 67, no. 12, pp. 5200–5210, Dec. 2020.
[20] A. Bytyn, R. Leupers, and G. Ascheid, "ConvAix: An application-specific instruction-set processor for the efficient acceleration of CNNs," *IEEE Open J. Circuits Syst.*, vol. 2, pp. 3–15, 2021.
[21] G. Ottavi, A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "A mixed-precision RISC-V processor for extreme-edge DNN inference," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2020, pp. 512–517.
[22] A. Amir et al., "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7388–7397.
[23] J. Sonet al., "Lip reading in the wild," in *Proc. Asian Conf. Comput. Vis.*, Cham, Springer International Publishing, 2016, pp. 87–103.
[24] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," in *Handbook of Systemic Autoimmune Diseases*. vol. 1, no. 4, 2009.
[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014, *arXiv:1409.1556*.
[26] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," Apr. 2017, *arXiv:1704.04861*.
[27] M. Davies, "Benchmarks for progress in neuromorphic computing," *Nature Mach. Intell.*, vol. 1, no. 9, pp. 386–388, 2019.
[28] A. Gruel, A. Vitale, J. Martinet, and M. Magno, "Neuromorphic event-based spatio-temporal attention using adaptive mechanisms," in *Proc. IEEE 4th Int. Conf. Artif. Intell. Circuits Syst.*, Incheon, Korea, 2022, pp. 379–382.
[29] A. Safa, I. Ocket, A. Bourdoux, H. Sahli, F. Catthoor, and G. G. E. Gielen, "Event camera data classification using spiking networks with spike-timing-dependent plasticity," in *Proc. Int. Joint Conf. Neural Netw.*, Padua, Italy, 2022, pp. 1–8.
[30] X. Xiao et al., "Dynamic vision sensor based gesture recognition using liquid state machine," in *Proc. Int. Conf. Artif. Neural Netw.*, Bristol, United kingdom, 2022, pp. 618–629.
[31] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H. -G. Stratigopoulos, "Neuron fault tolerance in spiking neural networks," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2021, pp. 743–748.
[32] J. Tang et al., "Relaxation LIF: A gradient-based spiking neuron for direct training deep spiking neural networks," *Neurocomputing*, vol. 501, pp. 499–513, 2022.
[33] N. Russo, H. Huang, and K. Nikolic, "Live demonstration: Neuromorphic robot goalie controlled by spiking neural network," in *Proc. IEEE Biomed. Circuits Syst. Conf.*, Taipei, Taiwan, 2022, pp. 249–249.
[34] S. U. Innocenti, F. Becattini, F. Pernici, and A. Del Bimbo, "Temporal binary representation for event-based action recognition," in *Proc. 25th Int. Conf. Pattern Recognit.*, Milan, Italy, 2020, pp. 10426–10432.
[35] J. Chen, J. Meng, X. Wang, and J. Yuan, "Dynamic graph CNN for event-camera based gesture recognition," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2020, pp. 1–5.

[36] Z. Wu, H. Zhang, Y. Lin, G. Li, M. Wang, and Y. Tang, "LIAF-Net: Leaky integrate and analog fire network for lightweight and efficient spatiotemporal information processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6249–6262, Nov. 2022.

[37] L. Cordone, B. Miramond, and S. Ferrante, "Learning from event cameras with sparse spiking convolutional neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, Shenzhen, China, 2021, pp. 1–8.

[38] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 2641–2651.

[39] M. P. R. Löhr and H. Neumann, "Contrast detection in event-streams from dynamic vision sensors with fixational eye movements," in *Proc. IEEE Int. Symp. Circuits Syst. Soc.*, Florence, Italy, 2018, pp. 1–5.

[40] J. Yang et al., "Modeling point clouds with self-attention and gumbel subset sampling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 3318–3327.

[41] B. Rueckauer et al., "NxTF: An API and compiler for deep spiking neural networks on intel loihi," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, pp. 1–22, 2022.

[42] S. -Y. Sun, H. Xu, J. Li, Q. Li, and H. Liu, "Cascaded architecture for memristor crossbar array based larger-scale neuromorphic computing," *IEEE Access*, vol. 7, pp. 61679–61688, 2019.

[43] A. Shrestha, H. Fang, D. P. Rider, Z. Mei, and Q. Qiu, "In-hardware learning of multilayer spiking neural networks on a neuromorphic processor," in *Proc. ACM/IEEE 58th Des. Automat. Conf.*, San Francisco, CA, USA, 2021, pp. 367–372, doi: 10.1109/DAC18074.2021.9586323.

[44] C. Zou et al., "Modular building blocks for mapping spiking neural networks onto a programmable neuromorphic processor," *Microelectronics J.*, vol. 129, 2022, Art. no. 105612.

[45] E. Ceolini et al., "Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing," *Front. Neurosci.*, vol. 14, 2020, Art. no. 637.

[46] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, USA, 2005, p. 41.

[47] M. Thompson et al., "Core-V verification environment," OpenHW Group, Tech. Rep., 2020. [Online]. Available: ttps://docs.openhwgroup.org/projects/core-v-verif/en/latest/corev_env.html

[48] M. Stimberg et al., "Equation-oriented specification of neural models for simulations," *Front. Neuroinform.*, vol. 8, 2014. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fninf.2014.00006

[49] W. Fang et al., Spikingjelly, 2020. [Online]. Available: https://github.com/fangwei123456/spikingjelly

[50] C. Frenkel, J. -D. Legat, and D. Bol, "MorphIC: A 65-nm 738k-synapse/mm2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 5, pp. 999–1010, Oct. 2019.

[51] B. Wang et al., "Shenjing: A low power reconfigurable neuromorphic accelerator with partial-sum and spike networks-on-chip," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2020, pp. 240–245.

[52] L. Wang et al., "LSMCore: A 69k-synapse/mm2 single-core digital neuromorphic processor for liquid state machine," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 69, no. 5, pp. 1976–1989, May 2022.

[53] C. Frenkel, M. Lefebvre, J. -D. Legat, and D. Bol, "A 0.086-mm 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.

[54] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 1947–1950.

[55] C. Mayr et al., "A biological-realtime neuromorphic system in 28nm CMOS using low-leakage switched capacitor circuits," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 1, pp. 243–254, Feb. 2016.

[56] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640m pixel/s 3.65mw sparse event-driven neuromorphic object recognition processor with on-chip learning," in *Proc. Symp. VLSI Circuits*, 2015, pp. C50–C51.

[57] H. Christopher et al., "VEGAS: Soft vector processor with scratchpad memory," in *Proc. 19th ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2011, pp. 15–24.

[58] L. Li, L. Gao, and J. Xue, "Memory coloring: A compiler approach for scratchpad memory management," in *Proc. 14th Int. Conf. Parallel Architectures Compilation Techn.*, St Louis, MO, USA, 2005, pp. 329–338.

[59] Z. Kang, L. Wang, S. Guo, R. Gong, Y. Deng, and Q. Dou, "ASIE: An asynchronous SNN inference engine for AER events processing," in *Proc. IEEE 25th Int. Symp. Asynchronous Circuits Syst.*, 2019, pp. 48–57.

[60] B. Su et al., "DCP: Improving the throughput of asynchronous pipeline by dual control path," in *Proc. IEEE 10th Int. Conf. High Perform. Comput. Commun. IEEE Int. Conf. Embedded Ubiquitous Comput.*, 2013, pp. 230–237.

[61] S. Li et al., "A multi-objective LSM/NOC architecture co-design framework," *J. Syst. Archit.*, vol. 116, 2021, Art. no. 102154.

[62] Z. Kang et al., "Hardware-aware liquid state machine generation for 2D/3D network-on-chip platforms," *J. Syst. Archit.*, vol. 124, 2022, Art. no. 102429.

[63] Y. Qiu et al., "A novel ring-based small-world NoC for neuromorphic processor," in *Proc. IEEE 32nd Int. Conf. Appl.-Specific Syst., Architectures Processors*, NJ, USA, 2021, pp. 234–241.

[64] Z. Kang et al., "Application-specific network-on-chip design space exploration framework for neuromorphic processor," in *Proc. 17th ACM Int. Conf. Comput. Front.*, Catania, Italy, 2020, pp. 71–80.

[65] S. Li et al., "SNEAP: A fast and efficient toolchain for mapping large-scale spiking neural network onto NoC-based neuromorphic platform," in *Proc. Great Lakes Symp. VLSI*, 2020, pp. 9–14.

[66] A. Hashmi et al., "A case for neuromorphic ISAs," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 145–158, Mar. 2011.

Zhijie Yang received the BEng degree in network engineering, and the MEng degree in computer technology from the National University of Defense Technology, Changsha, China, in 2018 and 2020. He is currently working toward the PhD degree in computer science and technology with the College of Computer Science and Technology, National University of Defense Technology, Changsha, China. His research interests include computer architecture and neuromorphic computing.

Lei Wang received the BEng and PhD degrees from the National University of Defense Technology, in 2000 and 2006, respectively. She has been assistant professor and associate professor with the College of Computer Science and Technology, National University of Defense Technology from 2007–2022. She is currently a professor with the Defense Innovation Institute, Academy of Military Sciences. Her current research interests include computer architecture, neuromorphic computing, asynchronous circuit, and artificial intelligence.

Wei Shi is currently associate professor with the College of Computer Science and Technology, National University of Defense Technology. His research interests include computer architecture, microprocessor design and information security.

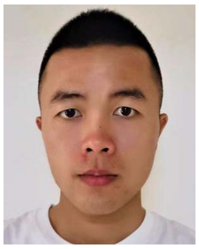
Yao Wang is currently assistant professor with the College of Computer Science and Technology, National University of Defense Technology. His research interests include VLSI, microprocessor design & implementation, and semiconductor devices & reliability.

Junbo Tie received the BEng and PhD degrees from the National University of Defense Technology, in 2013 and 2018, respectively. He is currently an associate professor with the College of Computer Science and Technology, National University of Defense Technology. He current research interests include computer architecture, artificial intelligence, and neuromorphic computation.

Feng Wang received the BEng, MEng, and PhD degrees in computer science and engineering from the National University of Defense Technology, in 2000, 2002 and 2013, respectively. He is a professor with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. His research interests include programming languages, compiler technology, programming models, parallel computing and run-time systems.

Xiang Yu received the bachelor's degree from the Dalian University of Technology, in 2020. He is currently working toward the postgraduate degree with the College of Computer Science and Technology, National University of Defense Technology. His current research interests include neuromorphic computing, RISC-V ISA and microarchitecture designs.

Linghui Peng received the bachelor's degree from the Wuhan University of Science and Technology, in 2019. He is currently an engineer with the College of Computer Science and Technology, National University of Defense Technology. His current research interests include neuromorphic computing, RISC-V ISA architecture and network on chip.



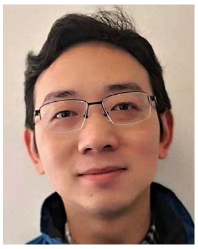
Chao Xiao received the BEng degree in software engineering and MEng degree from the National University of Defense Technology, Changsha, China, in 2020 and 2023 respectively. He is currently working toward the PhD degree in electronic science and technology with the College of Computer Science and Technology, National University of Defense Technology, Changsha, China. His research interests include computer architecture and neuromorphic computing.

Xun Xiao received the bachelor's degree from the Chongqing University of Posts and Telecommunications, in 2020, and the master's degree from the National University of Defense Technology, in 2023. He is currently working toward the PhD degree with the College of Computer Science and Technology, National University of Defense Technology. His current research interests include neuromorphic computing, spiking neural networks and dynamic vision sensor.

Yao Yao is currently working toward the graduate degree with the College of Computer Science and Electronic Engineering, Hunan University. His current research interests computer software toolchain.

Gan Zhou received the bachelor's degree from Hunan Normal University, in 2020. He is currently an engineer with the College of Computer Science and Technology, National University of Defense Technology. His current research interests include neuromorphic computing, RISC-V ISA architecture and network on chip.

Xuhu Yu received the bachelor's degree from the National University of Defense Technology, in 2020. He is currently working toward the postgraduate degree with the School of Computer Science and Technology, National University of Defense Technology. His current research interests include neuromorphic computing, spiking neural networks, and lipreading algorithms.



Rui Gong received the PhD degree from the National University of Defense Technology, in 2008. He is an associate professor with the College of Computer Science and Technology, National University of Defense Technology. His research interests include computer architecture, microprocessor design, information security and neuromorphic computing.

Xia Zhao (Member, IEEE) received the PhD degree in computer science and engineering from Ghent University, in 2019. He currently is an assistant professor with the National Innovation Institute of Defense Technology, China. His research interests include GPGPU architecture in general, and multi-program execution, cache hierarchy optimization and Network-on-Chip (NoC) design more in particular. He has served as a member of the External Review Committee of the leading computer architecture conferences ISCA and MICRO.

Yuhua Tang received the BEng and MEng degrees from the Department of Computer Science, National University of Defense Technology, China, in 1983 and 1986, respectively. She is currently a professor with the State Key Laboratory of High Performance Computing, National University of Defense Technology. Her research interests include supercomputer architecture and core router's design.

Weixia Xu received the BEng degree from the Nanjing University of Science and Technology, in 1984, and the MEng and PhD degrees from the National University of Defense Technology, in 1993 and 2018, respectively. He is currently a professor with the College of Computer Science and Technology, National University of Defense Technology. His current research interests include computer architecture, high performance microprocessor design, artificial intelligence, and neuromorphic computation.