JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 30, Number 9, 2023

Pp. 951–960
DOI: 10.1089/cmb.2023.0098

Research Articles

SWsnn: A Novel Simulator for Spiking Neural Networks

ZHICHAO WANG,[1,2] XUELEI LI,[1] JIANPING FAN,[3] JINTAO MENG,[1] ZHENLI LIN,[4] YI PAN,[1] and YANJIE WEI[1]

ABSTRACT

Spiking neural network (SNN) simulators play an important role in neural system modeling and brain function research. They can help scientists reproduce and explore neuronal activities in brain regions, neuroscience, brain-like computing, and other fields and can also be applied to artificial intelligence, machine learning, and other fields. At present, many simulators using central processing unit (CPU) or graphics processing unit (GPU) have been developed. However, due to the randomness of connections between neurons and spiking events in SNN simulation, this causes a lot of memory access time. To alleviate this problem, we developed an SNN simulator SWsnn based on the new Sunway SW26010pro processor. The SW26010pro processor consists of six core groups, each with 16 MB of local data memory (LDM). LDM has the characteristics of high-speed read and write, which is suitable for performing simulation tasks similar to SNNs. Experimental results show that SWsnn runs faster than other mainstream GPU-based simulators when simulating a certain scale of neural network, showing a strong performance advantage. To conduct larger scale simulations, SWsnn designed a simulation computation based on a large shared model of Sunway processor and developed a multiprocessor version of SWsnn based on this mode, achieving larger scale SNN simulations.

Keywords: simulation, SW26010pro, spiking neural network.

1. INTRODUCTION

A FAST, RESOURCE-EFFICIENT, and user-friendly spiking neural network (SNN) simulator can facilitate neuroscientists to simulate the functions of some brain regions on high-performance clusters. They study brain science by reproducing biological neurons and their electrical activities on neural networks based on their understanding of known biological neural networks.

[1]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China.
[2]Southern University of Science and Technology, Shenzhen, China.
[3]University of Chinese Academy of Sciences, Beijing, China.
[4]Shenzhen University General Hospital, China.

The current classic mainstream SNN simulators mainly include Nest (Gewaltig and Diesmann, 2007), Brian (Goodman and Brette, 2009), and Neuron (Haberly, 2001). These SNN simulators were developed earlier and are constantly being iteratively updated. They provide a relatively friendly interface, but they do not have much advantage in performance. Based on them, some software such as Brian2 (Stimberg et al., 2019) and Brian2genn (Stimberg et al., 2020) have also improved on this basis, hoping to achieve better performance or make the simulator more convenient for users to use.

With the development of graphics processing unit (GPU) technology, many simulators are designed based on GPU, and they can often achieve better performance. GeNN (Yavuz et al., 2016) uses code generation to accelerate computation based on Nvidia GPU, while its new generation version GeNN uses a ''program connectivity'' simulation method, where connectivity and synaptic weights are generated ''on-the-fly.'' However, it is difficult to generate synaptic weights and connectivity when simulating plasticity synapse models.

CARLsim (Niedermeier et al., 2022) is an open-source SNN simulator written in C++, which mainly supports Izhikevich spiking neuron model. It can use central processing unit (CPU) and GPU for simulation. It has also undergone many version updates. In addition, there are NeuronGPU (Golosio et al., 2020), Spice (Bautembach et al., 2021), and other simulators based on GPU development, which also achieve good performance and are easy for users to use.

In SNN simulators, the synaptic connections between neurons have randomness and sparsity, as well as the uncertainty and randomness of spike event occurrence frequency in neurons, which makes memory access expensive. To achieve better simulator performance, a new simulator SWsnn is designed on the SW26010pro processor. The SW26010pro processor has 16 MB of local data memory (LDM) in each core group (CG), which can accommodate enough data. The read and write speed of LDM is fast, which is suitable for SNN simulation. In Section 2, we will introduce the simulator SWsnn as well as the SW26010pro processor and the design process of SNN simulator. Section 3 introduces the parallelization design. In Section 4, we will show the performance of SNN simulator on SWsnn. Section 5 is conclusion and discussion.

2. SWSNN AND DESIGN

2.1. SWsnn

The SNN simulator SWsnn is designed based on the new generation of the Sunway architecture, which is convenient for neuroscientists to use and has better performance than other simulators to some extent. The Sunway processor SW26010pro main core supports C/C++ language, the slave cores supports C language, and the simulator SWsnn is mainly written in C/C++ language. The simulator SWsnn supports mainstream neuron leaky integrate and fire (LIF) models (Dayan et al., 2003) and Izhikevich (Izhikevich, 2003) model, as well as mainstream synapse models. It uses a clock-driven simulation algorithm. It supports arbitrary size of time step and allows setting time parameters by oneself. It also supports parallel computing with multiple processors. In addition, the SWsnn simulator does not require too much additional compilation time, nor does it take too long to initialize the neural network. The operation is relatively simple and can be used effectively.

2.2. SW26010pro

SW26010pro is an upgraded version of SW26010 processor, which has a significant improvement in performance compared with the previous one. SW26010pro processor is a heterogeneous multicore architecture, consisting of 6 CGs, each CG contains a manager processing element (MPE) and 64 computing processing elements (CPE). MPE is also called the main core, and CPE is called the slave cores. The MPE mainly functions as control, communication, IO, and so on, and the CPE is mainly used for computation. Most of the computing power of SW26010pro processor is concentrated on the CPE.

SW26010pro has a remarkable advantage: each slave core has 256 KB of LDM, which means that each CG has a total of 16 MB of LDM. LDM has fast read and write speed, so it can avoid too much time consumption caused by memory access. The large-capacity LDM feature makes SW26010pro suitable for the SNN algorithm. Many applications have achieved good results with the help of Sunway (Li et al., 2021). Figure 1 shows the structure diagram of SW26010pro.

2.3. Initialize neural network

For the simulation of simple neurons, it is only necessary to iterate according to the mathematical formula of the neuron model to determine whether to send and transmit spike events. But simulating large-

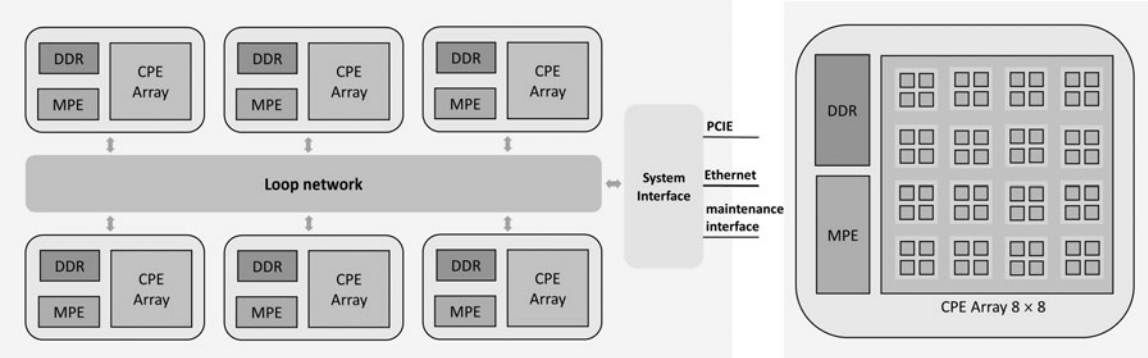

FIG. 1. SW26010pro processor. CPE, computing processing elements; MPE, manager processing element; PCIE, peripheral component interconnect express.

scale SNNs requires more complex factors. According to the characteristics of SW26010pro processor and SNN, we divide the simulation process into three important parts: initialize neural network, neuron update, and neural network update Figure 2 shows the entire simulation process. In the initialization process, allocate corresponding neurons and synapses memory space on SW26010pro processor and initialize parameters such as neurons and synapses parameters and connection methods between neurons according to requirements. The neuron network update part mainly refers to solving neuron state variables, that is, using methods such as Runge–Kutta or Euler to solve differential iteration equations of neurons. Since slave cores have most of computing performance of CG, this part of neuron solving is undertaken by slave cores. The neural network update part mainly refers to reading and processing synapses part and inputting data required by neurons.

In SNN simulation, two types of data need to be stored and processed: neurons and synapses. Synapse data occupy most of the storage space, and the number of synapses is thousands of times that of neurons. Although different types of neurons and synapses occupy different storage spaces, the difference is not large. However, due to the number of synapses, the space occupied by synapses is nearly a thousand times that of neurons. Therefore, we put neuron information in LDM and synapse information in main memory.

As shown in Figure 3, the left side of Figure 3 represents neuron storage, and the adjacency table on the right represents synapse storage. Each neuron is connected with synapses, and the number of synapses

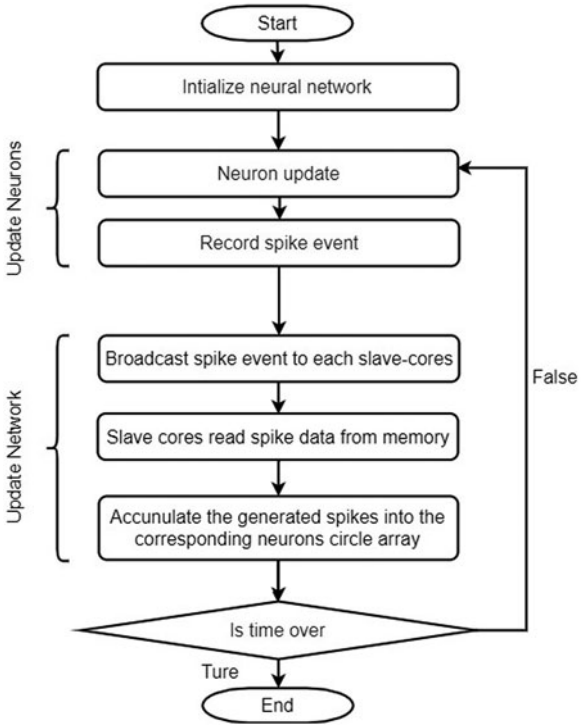

FIG. 2. Spiking neural network simulation process.

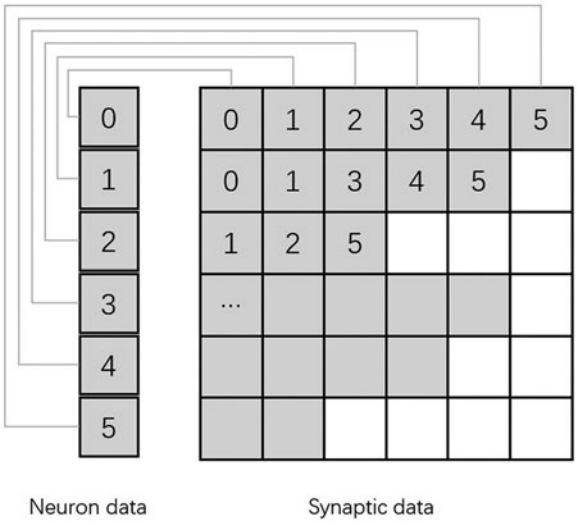

FIG. 3. Memory distribution.

connected by different neurons is also different. Each neuron connects thousands of synapses, and each synapse corresponds to a pre-neuron and a post-neuron. We set the storage of synapses as a two-dimensional structure array of structures. The rows are arranged according to the order of post-neurons, and the columns are arranged according to the order of pre-neurons.

In Figure 3, the neuron numbered 0 in the first row corresponds to neurons 0, 1, 2, 3, 4, and 5. In SNNs, synapses of neurons can be connected to themselves. Here, we call neuron 0 as the pre-neuron and neurons 0, 1, 2, 3, 4, and 5 as post-neurons. When neuron 0 generates a spike signal, it sends a spike signal to other connected neurons.

2.4. Neuron update

The neuron update part refers to the iterative calculation of the differential equations of all neuron voltages and currents. We use different neurons and corresponding mathematical formulas to calculate according to the requirements of neural network construction. As known from the memory distribution in the previous section, neuron data are stored in LDM in the slave cores. Although different types of neurons have different parameter settings, they are essentially solving for membrane voltage and current.

In the actual calculation process, we set the time step according to the requirements. Neuron updates are performed at each time step. In the neuron calculation process, neurons read currents from other neurons transmitted by synapses and then calculate membrane voltages according to neuron internal parameters and other information. However, other neurons may send spike signals to target neurons at each time step, and it takes time for spikes to be transmitted, which is also called delay. That is, neurons need to calculate spike information sent before time t and transmitted after a period of time at time t.

To deal with this situation, we set a circular array behind each neuron to store current information from other neurons. When neurons are stimulated before a period of time before time t, they have accumulated

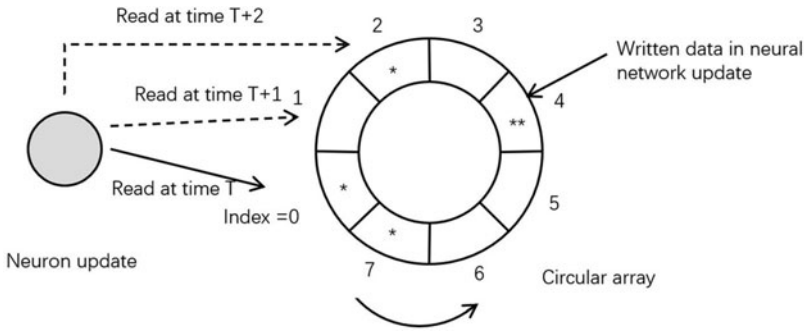

FIG. 4. Circular array.

current signals on the circular array according to delay and pre-neuron stimulation time. When time changes, the circular array pointer moves accordingly. The whole process is shown in Figure 4. As time changes, the circular array rotates counterclockwise, and the previously read data are cleared until the next write. As the time step changes, the circular array rotates counterclockwise, and each step of neuron update requires reading the current input from other neurons.

2.5. Neural network update

The network update part refers to reading weight and delay information from synapses, and adding current to the neurons connected by them according to the collected information of stimulated neurons, to facilitate subsequent calculations. In the previous process about neuron update, we created a structure array to store the neuron number and corresponding stimulation time of each time step. After that, the slave cores need to traverse all neurons that send spikes and perform spike signal transmission.

Since neuron information is stored in LDM space of their own slave cores, LDM space inside slave cores is relatively independent, and information between slave cores is not shared. In this process, each slave core needs to know the sending information of other neurons, which requires synchronization. We use a method called direct memory access (DMA) to perform calculations. DMA is a function for batch data transfer between main memory and LDM, which can only be initiated by slave cores for information read and write functions. DMA is an efficient and low-latency memory read and write method. We use the DMA method to first transfer the neuron number, time, and other information of sending spikes to the main core and then synchronize each slave core to read the main core's information in batches into the slave cores, which can achieve better results.

When all neurons on the slave cores obtain spike information from stimulated neurons, the slave cores read back corresponding synapse information into LDM space of slave cores by the DMA method. And according to delay, accumulate to different positions of post-neuron circular array, to facilitate neuron update calculation of next time step.

Biologically, a neuron becomes unresponsive to further stimuli during the refractory period, which lasts until the membrane potential recovers to the resting level after generating an action potential. This implies that a neuron cannot fire multiple spikes within a certain time interval. The LIF neuron model is relatively simple and sets the refractory period to 2 ms. In the clock-driven simulation of SNNs, the simulation resolution is usually 0.1 ms or smaller. Moreover, the spike transmission between neurons involves a certain delay, which follows an interval distribution. The minimum delay value is typically around 1 ms, depending on the specific network design. Therefore, a neuron will not receive repeated inputs within a short time span, and the network update can be performed less frequently than the neuron update. The spike transmission within the minimum delay will not affect the experimental results significantly. Hence, the network update can be done after several neuron updates.

3. PARALLEL DESIGN

Since neurons and synapses have high memory occupancy and computation, a single computing CG often cannot support such tasks when performing large-scale brain region simulations. To expand the simulation scale and improve the simulation efficiency, we adopt a parallel strategy. Specifically, we mainly use two parallel simulation methods, namely large shared mode and multiprocessor mode.

3.1. Large shared mode

SW26010pro provides a parallel mode known as large shared mode. In this mode, a single main core can manage the slave cores of multiple CGs within the current processor for calculation. This means that only the main core of a single CG is utilized for task dispatch and allocation work while all 384 slave cores within the processor can be used to perform calculation tasks. The use of large shared mode does not differ significantly from normal task dispatching between main and slave cores of a single CG. The only difference lies in the function interface called and there is no need for significant changes to the entire simulation process.

In large shared mode, a single CG can access the entire processor memory by submitting the instruction of the task. With 16 GB of memory space for a single CG and six CGs adding up to 96 GB of main memory space, this undoubtedly increases the scale of simulation. Utilizing this method allows for full use of the

high bandwidth and low latency inside the processor to improve both the accuracy and real-time performance of simulation. Furthermore, simulating an SNN based on processors as basic units yields better results than simulating based on CGs as basic units. This approach reduces the difficulty of memory allocation and lowers communication costs between CGs.

3.2. Multiprocessor parallelism

During the process of multiprocessor simulation, it is necessary to redesign both the memory structure and simulation process. The first step involves reallocating memory. The entire neural network must be divided into as many disjoint neural networks as possible to enable relatively independent operation and minimize information transmission between different processors. Storage of the entire SNN primarily consists of neuron storage and synapse storage. Neuron information and synapse information must be stored on different processors, respectively.

The memory of neurons is evenly distributed to different processors to balance computing power, whereas the memory allocation of synapses is more complex, mainly due to the fact that the storage information of synapses involves the number of post-neurons connected by pre-neurons. When initializing synapse information, the number of post-neurons connected by synapses in synapse information is random, but for subsequent convenience of calculation, it needs to be sorted according to the number of post-neurons. Moreover, when performing multiprocessor parallel computing, to reduce communication between processors, it is necessary to split synapse data as shown in Figure 5.

Synapses whose connected post-neurons are located in processor A are lightly shaded, and synapses whose connected post-neurons are located in processor B are darker. We split synapse data according to this, thereby reducing communication costs between processors. There is no difference between neuron update part and previous introduction. Iterative calculation of mathematical formulas is performed on slave cores to obtain neuron spike information. Each processor obtains the spiked neuron's number and the spiked time of neurons on that processor. At this time, each processor does not know other processors' neuron spike situation. So, communication between different processors is required to implement the next step of the neural network simulation process.

SWsnn adds a synchronization process between neuron update and neural network update. In neuron update part, each processor records number and time of neurons sending spikes. In synchronization process, we use the message passing interface (MPI) method for interprocessor information synchronization. Each processor needs to obtain all processors' spike signal information at this moment.

After synchronization between different processors, processor enters neural network update step. Because SNN has been split, computation between processors is relatively independent. Each processor traverses synapse information in main memory according to stimulated neuron information. In the main memory synapse storage structure, all the post-neurons connected by pre-neurons are located in this processor. Therefore, the cyclic iteration between neurons does not affect the entire simulation process and achieves good parallelism.

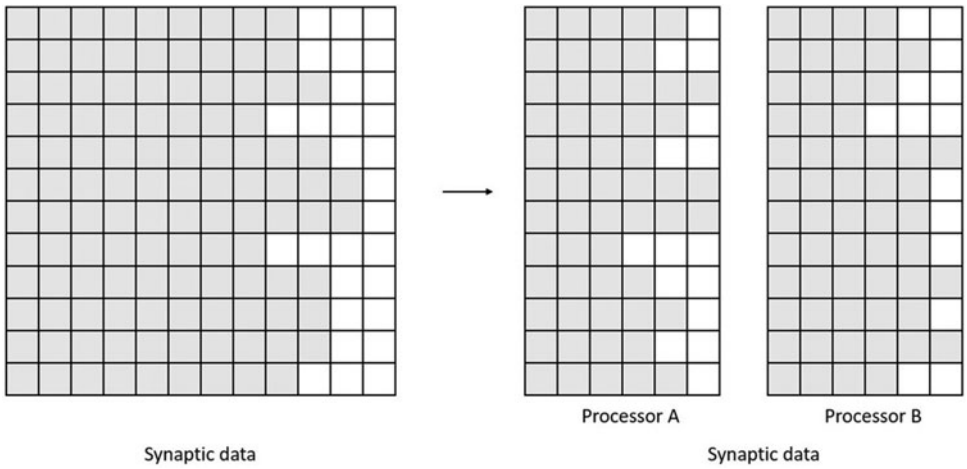

FIG. 5. Memory allocation.

4. RESULTS

4.1. Biological model

SWsnn provides a computing platform that can use SNNs to study the structure and computation of the cerebral cortex under realistic biological constraints. However, in reality, the structure of brain regions requires complex mathematical models to be modeled.

Inspired by the anatomical structure of the mammalian cortex, Potjans and Diesmann (2014) constructed a network model of the local cortical column microcircuit model and obtained the mathematical calculation model of the entire network based on the anatomical and physiological maps of the cortical column. The model uses LIF neurons for simulation, and the model is divided into four layers called L2/3, L4, L5, and L6. Each layer of neurons is divided into excitatory neurons and inhibitory neurons. There is also a Poisson input device outside the entire model to represent the brain's input stimulus to the entire column.

We ported the entire model to run on SWsnn SNN simulator. The parameter settings of the cortical microcircuit model on SWsnn are consistent with the original parameters in the article. In the process of model design, some parameters use random numbers as part of their values, so there are slight differences in simulation results, but they show consistent content. The raster plot of neurons stimulated on SWsnn is shown in Figure 6b, where each point represents that neuron with that number has a spike signal at this moment. The content shown by raster plot is consistent with original article experiment figure. We show histograms of firing rates for different types of neurons in different layers in Figure 6c. Their firing rates are also similar to original experiment. Due to random values design of neuron synapses and excitation current, firing rate error of different layers is around 4%, which is reasonable.

4.2. Single-processor performance analysis

GeNN is one of the fastest mainstream SNN simulators currently available. It uses Nvidia's GPU for simulation. This section compares experiments using SWsnn and GeNN software.

We use the large shared mode in SWsnn here, which uses a single main core to call the computing resources in the entire processor. The peak performance of SW26010pro processor is 13.2 TFLOPS. GeNN uses NVIDIA A100 PCIe processor for simulation under Linux environment. We use 40 GB of video memory, and A100's single-precision peak performance is 19.5 TFLOPS.

In this article, we use a relatively simple neural network simulation to do experimental comparison. The network consists of Izhikevich neurons; each neuron connects about 1000 neurons, which is similar to real biological model. We use external current input to replace the influence of other neurons on this population, thereby affecting the firing rate of the entire neuron population. We control the population firing rate at around 10 Hz by external excitation current size and set the simulation time step to 0.1 ms. In experiment, we also uniformly use Runge–Kutta fourth-order method for iterative calculation. In experiment, we set different neural network scales to perform multiple simulation calculations on SWsnn and GeNN.

As shown in Figure 7, we performed simulations on SWsnn and GeNN by setting different network sizes. Light represents the simulation time required on GeNN software, and dark represents SWsnn software's simulation time. It can be seen that with change of neural network simulation scale on two software, SNN's

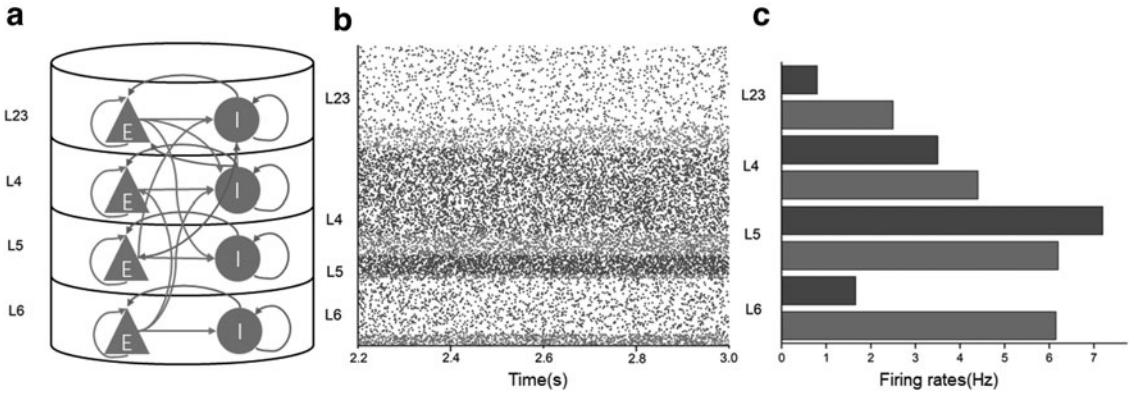

FIG. 6. Cortical column microcircuit model. **(a)** Schematic diagram of cortical column simulation. **(b)** Fixing grid diagram of neurons in different layers. **(c)** Excitation frequency diagram between different layers.

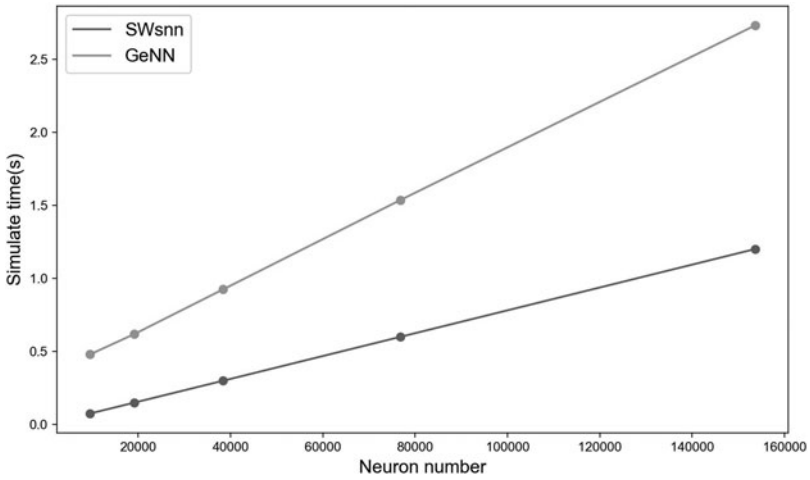

FIG. 7. Performance comparison between GeNN and SWsnn.

simulation time also increases linearly. But because SWsnn and GeNN use different processors and belong to different architectures, comparison between them is not perfect, so in experiment process, we converted GeNN's simulation time according to single-precision peak performance of SW26010pro processor and A100 processor. But this process can also reflect the performance of SWsnn. Under some same scale of SNN experiment, SWsnn has faster simulation speed.

As shown in Figure 8, we performed a more in-depth analysis of SWsnn's performance using different connection modes for neural networks. We controlled neuronal populations' firing frequency at 10 Hz using external current input. We used a simulation time step of 0.1 ms and applied Runge–Kutta's fourth-order method for iterative calculation. But the way the two types of neural networks are connected is not the same. Figure 8a shows a schematic diagram where each neuron connects to 1000 neurons in the entire neural network; this result matches what Figure 8 shows. In contrast, Figure 8b shows that neurons connect with a probability of 0.1.

We timed neuron updates and neural network updates separately during simulation. The results show that both updates take roughly equal time and increase linearly as neural network size grows. Each core traverses all synaptic information and performs DMA reading during neural network updates; however, DMA reading rate does not increase linearly with reading volume—it reaches peak performance and then stabilizes. When the connection probability is 0.1, with the increase of simulation scale, the number of neurons increases linearly, whereas the number of synapses increases nonlinearly due to the connection probability, which leads to the nonlinear increase of neural network update time.

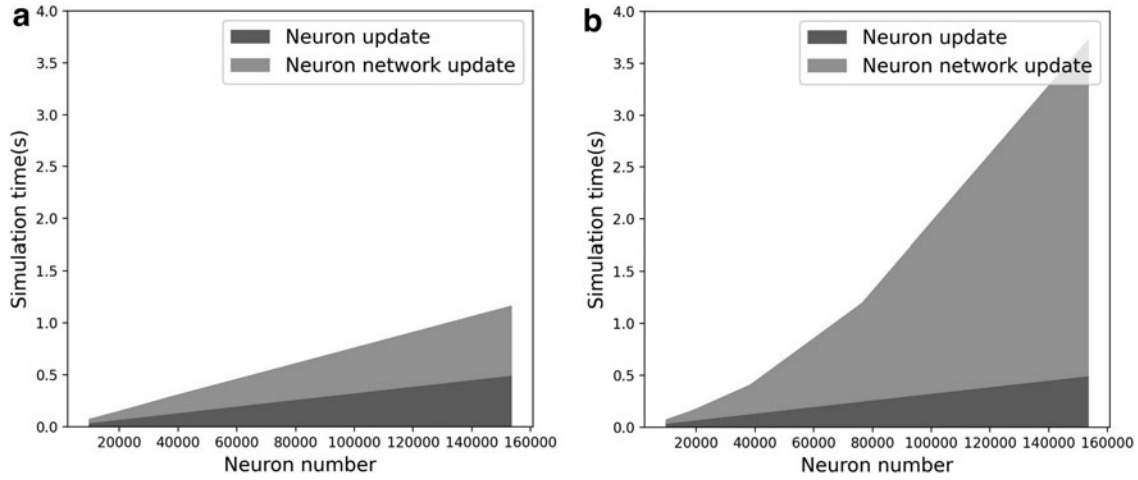

FIG. 8. Performance analysis graph of SWsnn. **(a)** Each neuron connects 1000 neurons. **(b)** Each neuron connects with a probability of 0.1.

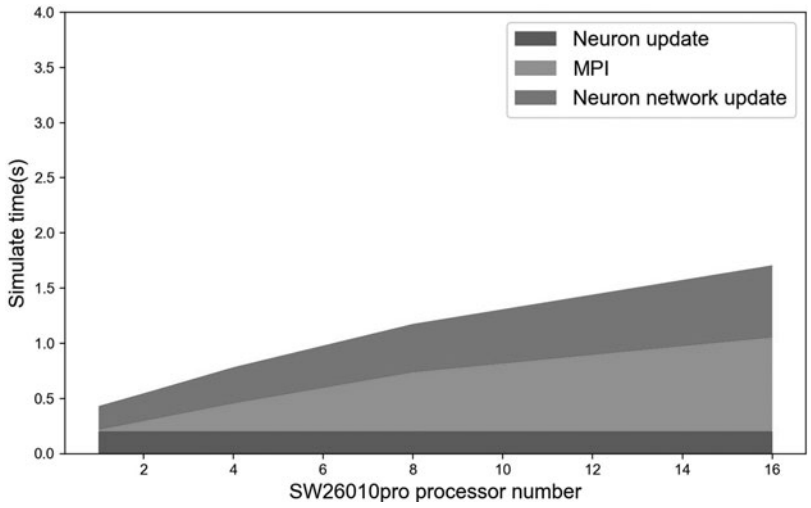

FIG. 9. Multiprocessor performance analysis.

4.3. Multiprocessor performance analysis

In this section, we test the parallelism of SWsnn. We use the large shared mode and take the processor as the basic computing unit for simulation.

The model used in experiment is similar to previous section. It is also through external current input to control neuron population's firing rate at 10 Hz, simulation time step is 0.1 ms, and also use Runge–Kutta fourth-order method for iterative calculation. This experiment is a weak scaling experiment. Each processor places 9600 neurons, and connection probability between neurons is 0.1.

As shown in Figure 9, with increase of processor number, total simulation scale increases, and time required for neuron update does not change, because number of neurons in each processor is fixed, and neuron update calculation amount will not change. However, with increase of simulation scale, MPI time increases significantly, reaching a fairly high level. In case of 16 processors, MPI time has exceeded sum of other two items. This is because in MPI communication process, each processor needs to know firing situation of other processors' neurons. With increase of processor number, this undoubtedly brings too much time consumption. Neural network update also increases with increase of simulation scale, which is also due to linear increase of synapse data with increase of simulation scale.

5. CONCLUSIONS

In this work, we propose an efficient SNN simulator SWsnn, which uses the SW26010pro processor to simulate SNNs. The simulator fully utilizes the processor features of Sunway SW26010pro and the characteristics of SNNs, achieving good performance. In addition, we use the large shared mode of SW26010pro processor to extend it. On this basis, we implement multiprocessor simulation by using MPI and improve parallelism as much as possible.

ACKNOWLEDGMENTS

We thank the funding support by the Youth Innovation Promotion Association (Y2021101), CAS to Y.W. And, we would also like to thank the SW26010pro processor provided by Sunway supercomputer.

AUTHOR DISCLOSURE STATEMENT

The authors declare they have no conflicting financial interests.

FUNDING INFORMATION

This work was partly supported by the Key Research and Development Project of Guangdong Province under Grant No. 2021B0101310002; National Key Research and Development Program of China Grant No. 2021YFF1200104; Strategic Priority CAS Project XDB38050100; National Science Foundation of China under Grant No. 62272449; and the Shenzhen Basic Research Fund under Grants JCYJ20210324102007021, RCYX20200714114734194 and KQTD20200820113106007.

REFERENCES

Bautembach D, Oikonomidis I, Argyros A. Multi-GPU SNN simulation with static load balancing. In: 2021 International Joint Conference on Neural Networks (IJCNN). IEEE; 2021; pp. 1–8.

Dayan P, Abbott LF. Theoretical neuroscience: Computational and mathematical modeling of neural systems. J Cogn Neurosci 2003;15(1):154–155.

Gewaltig MO, Diesmann M. Nest (neural simulation tool). Scholarpedia 2007;2:1430.

Golosio B, Tiddia G, De Luca C, et al. A new gpu library for fast simulation of large-scale networks of spiking neurons. *arXiv preprint arXiv:2007.14236*, 2020.

Goodman DF, Brette R. The brian simulator. Front Neurosci 2009;1(2):192–197.

Haberly LB. Parallel-distributed processing in olfactory cortex: new insights from morphological and physiological analysis of neuronal circuitry. Chem Senses 2001;26(5):551–576.

Izhikevich E. Simple model of spiking neurons IEEE transactions on neural networks. 2003.

Li F, Liu X, Liu Y, et al. SW_Qsim: A minimize-memory quantum simulator with high-performance on a new sunway supercomputer. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. St. Louis, MO, USA. 2021; pp. 1–13.

Niedermeier L, Chen K, Xing J, et al. CARLsim 6: An open source library for large-scale, biologically detailed spiking neural network simulation. In: 2022 International Joint Conference on Neural Networks (IJCNN). IEEE; 2022; pp. 1–10.

Potjans TC, Diesmann, M. The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. Cereb Cortex 2014;24(3):785–806.

Stimberg M, Brette R, Goodman DF. Brian 2, an intuitive and efficient neural simulator. Elife 2019;8:e47314.

Stimberg M, Goodman DF, Nowotny T. Brian2GeNN: Accelerating spiking neural network simulations with graphics hardware. Sci Rep 2020;10(1):410.

Yavuz E, Turner J, Nowotny T. GeNN: A code generation framework for accelerated brain simulations. Sci Rep 2016;6(1):1–14.

Address correspondence to:
Prof. Xuelei Li
Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences
Shenzhen 518055
China

E-mail: xl.li@siat.ac.cn

Dr. Yanjie Wei
Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences
Shenzhen 518055
China

E-mail: yj.wei@siat.ac.cn