Simeuro: A Hybrid CPU-GPU Parallel Simulator for Neuromorphic Computing Chips

Huaipeng Zhang, Nhut-Minh Ho, Dogukan Yigit Polat, Peng Chen, Mohamed Wahib, Truong Thao Nguyen, Jintao Meng, Rick Siow Mong Goh, Satoshi Matsuoka, Tao Luo, and Weng-Fai Wong, *Senior Member, IEEE*

***Abstract*—With the success of deep learning, there have been numerous efforts to build hardware for it. One approach that is gaining momentum is neuromorphic computing with spiking neural networks (SNNs), which are multiplication-free and open the possibility of using analog computing via novel technologies. However, to design effective and efficient hardware for such architectures, a fast and accurate software simulator is key. This article presents Simeuro, a fast and scalable system-level simulator for SNN models used in neuromorphic accelerators. The simulator uses spike-level details and configurable architectural constraints that are independent of the underlying hardware implementation. Simeuro supports a wide range of features including analog computing, novel memory (currently, RRAM is supported), and a full network-on-chip. The simulator can provide detailed simulation results such as routing statistics, energy consumption, delay, and accuracy of arbitrarily defined SNN architectures. Our simulator leverages a CPU-GPU hybrid environment to expedite the simulation by scaling out to multi-nodes equipped with multi-GPUs. We are able to conduct core simulations for a system-scale SNN chip of 20,000 neuromorphic cores on up to 512 A100 GPUs in a few minutes.**

***Index Terms*—Neuromorphic computing, chip simulation, deep learning.**

Manuscript received 22 November 2022; revised 25 May 2023; accepted 24 June 2023. Date of publication 3 July 2023; date of current version 22 August 2023. This work was supported in part by the Singapore Government's Research, Innovation and Enterprise 2020 Plan (Advanced Manufacturing and Engineering domain) under Grants A1687b0033 and A1892b0026. This paper was based on results obtained from JPNP20006 project, commissioned by the New Energy and Industrial Technology Development Organization (NEDO). Recommended for acceptance by P. Bangalore. *(Corresponding author: Tao Luo.)*

Huaipeng Zhang, Rick Siow Mong Goh, and Tao Luo are with the Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore 138632 (e-mail: zhang_huaipeng@ihpc.a-star.edu.sg; gohsm@ihpc.a-star.edu.sg; tluo001@e.ntu.edu.sg).

Nhut-Minh Ho, Dogukan Yigit Polat, and Weng-Fai Wong are with the School of Computing, National University of Singapore, Singapore 119077 (e-mail: minhhn@comp.nus.edu.sg; yigit@u.nus.edu; wongwf@nus.edu.sg).

Mohamed Wahib and Satoshi Matsuoka are with the RIKEN Center for Computational Science, Kobe 650-0047, Japan, and also with the Tokyo Institute of Technology, Tokyo 152-8550, Japan (e-mail: mohamed.attia@riken.jp; matsu@acm.org).

Peng Chen is with the, and also with the National Institute of Advanced Industrial Science and Technology, Japan, RIKEN Center for Computational Science, Tokyo 100-8921, Japan (e-mail: chin.hou@aist.go.jp).

Truong Thao Nguyen is with the National Institute of Advanced Industrial Science and Technology, Tokyo 100-8921, Japan (e-mail: nguyen.truong@aist.go.jp).

Jintao Meng is with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 100045, China (e-mail: jt.meng@siat.ac.cn).

Link to access the simulator: https://github.com/huaipeng/ncc.simulator.exe

Digital Object Identifier 10.1109/TPDS.2023.3291795

I. Introduction

RECENT trends in machine learning and artificial intelligence have sparked a general interest in the design of specialized hardware for deep neural networks. Deep neural networks (DNNs) are widely used in many application domains including image classification [1], language translation [2], and multimedia-related tasks [3]. Although DNNs are very effective at complex inference tasks, for fast execution they typically run on hardware that relatively consumes too much energy, such as GPUs, TPUs, and digital ASICs. DNNs are, in fact, mathematical models that take inspiration from neuro-biological systems, such as the brains of mammals. However, DNNs contrast with natural neural networks in the way the data is processed. DNNs work with real-valued input and process it through non-linear activation functions. This is indeed a gross oversimplification of natural neural networks where inputs are sequences of electrical spikes and activation occurs when certain thresholds in electrical potential on membranes are reached. Neural network models that try to model this exact behavior also exist in the field of neuromorphic engineering. Such neural networks are called Spiking Neural Networks (SNNs) and have been proven to be as equally capable in inference as conventional DNNs [4]. However, our current computer systems are not suitable for efficient execution of such models. The need for better computational architectures for SNN inference motivated the research and development of specialized analog neuromorphic hardware that allows fast and power-efficient evaluation of SNNs [5], [6]. Research and development of neuromorphic hardware is a relatively new direction, and there is extensive ongoing work for enhancing currently available systems. Since hardware production is a costly and time-consuming process, simulators are crucial in the evaluation of newly developed methods to explore different configurations of neural models, architectures, hardware configurations, and materials. Detailed simulators prevent hardware production overhead from affecting the research and development process.

Previous work on neuromorphic chip simulators either does not provide low-level hardware configurations (crossbar type, material, and the associated noise model) or has insufficient scalability. In addition to system-level simulation, other simulators emulate only the neuromorphic core [7], [8] or spiking neuron behavior [9], [10], [11], [12]. This is not sufficient for a full representation of the neuromorphic chip. This motivates our work; we strive to achieve both high-detail simulation results

1045-9219 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

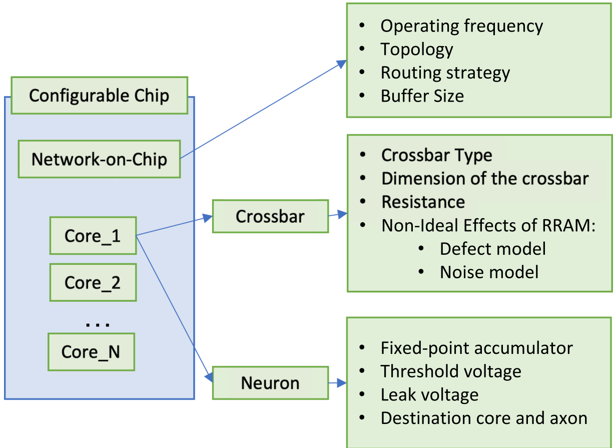


Fig. 1. Configurable parameters in Simeuro to simulate a neuromorphic chip with realistic hardware defects, neuron model, and different network topology.

and high scalability. We achieved this by using GPU-accelerated supercomputers and clusters to scale and accelerate certain parts of the simulation. We also use the otherwise idle CPUs by dynamically allocating tasks in the simulation either to the CPU or GPU, based on whether a specific task favors CPUs (i.e., latency-sensitive parts of the simulation) or GPUs (i.e., throughput-sensitive parts of the simulation).

In this paper, we present *Simeuro*, a system-level simulator capable of large-scale and fine-grained simulation of various neuromorphic chips running real workloads. The simulator is a necessary middleware for evaluating the performance of the neuromorphic chip and helping engineers debug the complex issues involved in mapping spiking neural networks onto chips. Simeuro is inspired by a multi-threaded single-CPU simulator by Lee et al. [13] w.r.t. to the features it can simulate. In contrast to the simulator proposed by Lee et al. Simeuro introduces qualitative, quantitative, and practicality aspects that could transform the landscape of neuromorphic simulations from small and simple proof-of-concept chips to be able to simulate current and next-generation neuromorphic chips at a very fine grain. Designing and implementing a simulator that scales to 1,000 s of GPUs or CPUs involves several challenges. First, it is imperative to orchestrate the distributed and hybrid simulation environment where different components of the simulation can scale (i.e., asynchronous stall-free pipeline). Second, the core simulation on the GPUs requires many optimizations to achieve high performance (e.g., shared memory blocking, altering the data access pattern and memory layout, etc.), Third, efficient parallelization of the routing component on the CPU requires a lock-free algorithm to simulate the fine-grained details of routing. Finally, an end-to-end approach is required to do a system-scale simulation starting from splitting the work and loading it, and up until the results are generated.

Simeuro can simulate a chip with a configurable NoC and support both the digital crossbar and the RRAM crossbar routing using fine-grained configurable parameters to describe each component of the chip in detail. Fig. 1 describes how we can configure the chip to simulate according to the defect models and the chip described in Section II-A. Besides the hardware model to simulate, the user can also selectively use GPUs and/or CPUs to speed up the simulation using our multi-GPU multi-node design described in Section IV. Simeuro can output cycle-accurate data on how the spikes are processed inside the chip and the power consumption of the chip. Thus, it can:

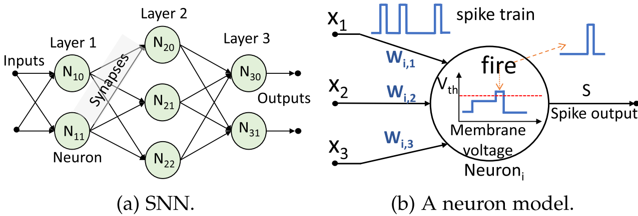
- measure the impact on accuracy when:
  - using a different material (e.g., Cu, HfO_2, NiO) for RRAM crossbar with a variety of noise and defect models;
  - using different number of bits for the digital crossbar to store quantized weights;
  - using different neuron models;
- analyze the network traffic of different 2D-mesh network topology, chip configuration, and workload;
- analyze the power consumption of the chip with a specific configuration and workload;
- simulate a physical chip with the configurations shown in Fig. 1.

Although Simeuro can simulate arbitrary chips given a configuration file and spike sources, it cannot simulate an SNN model directly because the neural network layers can have arbitrary dimensions and connections while the core has limited dimensions and connections due to the crossbar design (e.g., 256x256) and NoC topology. To simulate an SNN model from a software framework, the user has to map the network architecture to the real cores with limited dimensions and give the final result as input to Simeuro. The task is done using a separate tool [14]. This distinguishes it from other functional simulators that work on abstract SNN models that do not consider physical constraints arising from hardware implementation. Also, Simeuro being a functional simulator distinguishes it from low-level hardware description language (HDL) simulators that would be more accurate in simulating actual hardware but would take so long that simulating a realistic SNN model would simply be infeasible.

In summary, Simeuro advances the state-of-art in system-level neuromorphic simulations as follows:

- *Qualitatively*: It supports highly configurable simulation of digital chips that may use novel *resistive random access memory* (RRAM) technology,
- *Quantitatively:* The simulator is highly scalable. We have scaled simulations using up to 512 GPUs to simulate neuromorphic chips with up to 20,000 of neurosynaptic cores. The simulator can run in a hybrid fashion on both GPUs using CUDA and multi-threaded CPUs using OpenMP. Simeuro can maximize performance by allocating suitable tasks for GPUs and/or CPUs depending on a user-specified configuration as well as the underlying hardware. To speed up the simulation of the *network-on-chip* (NoC), we introduce lock-free parallelization that avoids race conditions during inter-router communication,
- *Practicality:* The simulator can run on Linux, Windows, and Mac OS. Simulations can be scaled to multi-nodes with multi-GPUs in a transparent way, and the modular design of the simulator components allows for extending the simulator with new features.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.



(a) SNN. (b) A neuron model.

Fig. 2. Example of a SNN and its neuron model.

II. Background and Motivation

A. Neuromorphic Computing

In this section, we will introduce neuromorphic computing and some related concepts.

1) Spiking Neural Networks: A Spiking Neural Network (SNN) is a class of artificial neural networks that mimic the actual biological behavior of the neuron. The model involves a network of connected neurons. Unlike conventional artificial neural networks where each neuron transmits information as real values at each time step, neurons in SNNs only fire when the accumulated effect to itself reaches a certain firing threshold. This is usually modeled as the accumulated membrane voltage of each neuron as shown in Fig. 2(a). One of the most popular neuron models is the *leaky-integrate-and-fire* (LIF) model. At each time step, the membrane voltage in the neuron will be updated after integrating the synaptic inputs (X) from all the axons and subtracting a pre-configured leak amount. When the membrane voltage in the neuron exceeds a pre-defined threshold (V_{th}), the neuron will fire a spike (S) and the voltage will be reset to the initial level. Fig. 2(b) demonstrates the integrate-and-fire process of the neuron i.

Each input's contribution towards the final firing of the target neuron is determined by its weight (W). The fired signal (S) will travel to its connected neuron and increase the chance for the target neuron to fire the subsequent signal to other neurons. This behavior is similar to the neuron firing model in the brain. Although we can simulate such behavior using software simulation, running SNN efficiently requires special hardware, which belongs to a new class of computers specifically designed for such tasks.

B. Hardware Model and Simulation

Neuromorphic chips are developed as AI hardware specialized in running SNNs. Real physical neuromorphic chips are becoming increasingly produced in both academic labs and industry, most notably IBM's TrueNorth [6] and Intel's Loihi [5]. Neuromorphic chips are typically designed to follow the structure of artificial neural networks: a large number of small computing cores correspond to a small batch of neurons whilst configuring connections to subsequent neurons. Neuromorphic chips with several thousands of cores can execute deep SNNs more efficiently than CPUs and GPUs, in terms of both compute time and energy.

C. Core Design: Crossbar

The basic computation of each layer in Fig. 2 can be mapped to crossbars [15] in neurosynaptic cores[1] which can connect the input to the actual output of each layer. In the crossbar, the horizontal lines (word lines) are connected to the input and receive input spikes. The vertical lines (bit lines) are connected to the output neurons. The crossing point between the two lines stores the synaptic weight information of each connection. We use two different types of crossbars, namely a *digital* and a *RRAM* based crossbar, to store weights and perform the multiply-accumulate operation on each vertical line. In the next two sections we elaborate on the merits and demerits of digital and RRAM crossbars.

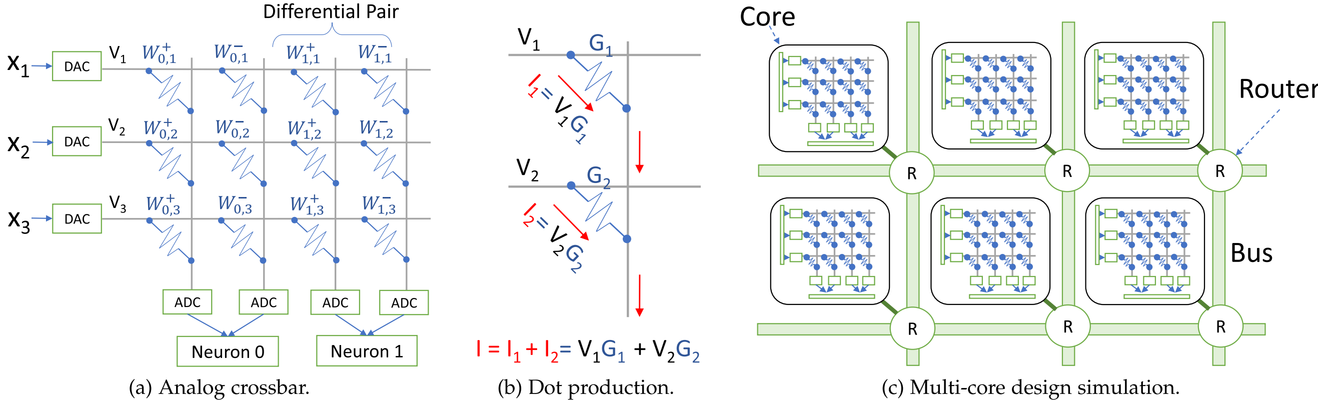
1) Digital Crossbar: This type of crossbar is used to store synaptic weights in digital form. The weights would then be read and used in the dot product with input values to compute the final output. The computation module is implemented using conventional digital circuitry to realize the multiply-and-accumulate operation. The data is typically quantized to reduce the bitwidth which in turn would lead to a reduction in memory and energy consumption [16].

2) RRAM Crossbar: To reduce energy usage, researchers have been utilizing analog systems to perform matrix-vector multiplication. The most common design is to use the conductance (G) of *resistive random access memory* (RRAM) cells to represent weights [17], [18], [19], [20]. RRAM is a novel form of non-volatile memory technology that varies the resistance of a special solid dielectric material to store data. The input value is represented as Voltage (V) value by using a *digital-to-analog converter* (DAC) as in Fig. 3(a). The electric current (I) after each cross point is the multiplied value of weight and input in analog domain in accordance to Ohm's Law. In the vertical line in Fig. 3(b), the current of each input is added together according to Kirchoff's Current Law. This is called a *current sum.* In effect, the entire vertical line performs a dot product between the inputs V and weights W to produce the output of each neuron. Assuming that we have inputs X_1, X_2 being converted to V_1, V_2, the 2 weights W_1, W_2 are represented by the conductance G_1, G_2, Fig. 3(b) shows how the dot product result can be represented by the current I at the end of each vertical line.

For each RRAM cell, the material used to design them and the operating environment affect the accuracy of the output values. RRAM switches by the formation and destruction of nanoscale conductive filaments in the dielectric material. This physical phenomenon is subject to noise. In addition, the movement of charge carriers inside the filament gives rise to *random telegraph noise* (RTN). We used the same RRAM model as Lee et al. [13] that also incorporated a noise model. The noise model is also described in [21]. Lee et al. implemented the method using a uniform random number generator on CPU. We used CURAND on the GPU to achieve a similar result.

[1]For the remainder of this paper we use "*core*" to mean neurosynaptic core unless otherwise stated.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.



(a) Analog crossbar. (b) Dot production. (c) Multi-core design simulation.

Fig. 3. Analog crossbar and Multi-core designed simulation. (a) Analog crossbar used to compute the multiplication of weights matrix ($W_{x,y}$) and axon inputs vector (X_i). (b) Dot product of Voltage (V_i) and Conductance (G_i) results in the value of current I at the end of each vertical line. (c) Multi-core design with the building block of each core as crossbar connects to the NOC. The routers choose the bus paths to send spikes to transfer them to their destination cores.

Since conductance is an unsigned value while weights are signed values, two vertical lines, one for the positive and another for the negative value, is needed for each neuron. This is called a *differential pair* [22]. The differential pair for $W_{i,j}$ is marked as $W^+_{i,j}$ and $W^-_{i,j}$ in Fig. 3 for the positive and the negative value, respectively. At each neuron, we combine the accumulated value of the differential pair together to obtain the final output.

D. Multi-Core Design and Communication Network

The crossbar in Fig. 3(a) and (b) can be used to process the dot product of an SNN layer or part of a layer. However, when designing a chip for handling larger neural networks of different architectures, it is not efficient to use a large crossbar in a single neurosynaptic core to process. The most popular method is to organize these computational crossbars into multiple neural cores. Depending on the workload, a number of cores will be allocated and used. Fig. 3(c) shows how to organize the crossbar in Fig. 3(a) to form a larger system that can have thousands of cores and process a very deep network having millions of neurons. Each core in Fig. 3(c) consists of a crossbar, an input module to decode and receive the spike, and an output module that executes the neuron's activation model to fire spikes. The neurosynaptic cores will connect to each other via an interconnecting network with a routing mechanism that routes spikes among neurosynaptic cores.

Because routing information and the spike outputs are dynamic and affect the total energy usage, the communication network is a vital part of the real chip and in our simulation. There are two main communication models supported by our simulator.

1) Ideal Network: The assumption of an ideal network means there would be no delay in transferring spikes between cores. An ideal network also assumes that each core is fully connected to all other cores. In practice, this network is not realistic, yet can be helpful in research and development when users focus only on the processing capability of cores and prefer to ignore the communication cost.

2) Topology-Based Network: There are several topologies that can be used to realize the communication network [23], [24], [25]. In our simulator, we support the mesh topology, as it is one of the most popular topologies used at the chip level. In a 2D mesh network (Fig. 3(c)), there is a router connected to each core that sends and retrieves spikes on the interconnecting bus. The router at each interconnect also has the responsibility to determine which direction to send each spike into, according to the destination core. The routing mechanism needs to be fully simulated for the development of a real large-scale chip.

E. Motivation for a Large Scale System-Level Simulator

Designing real neuromorphic chips is expensive. With various network architectures and sizes, the decision to choose the core size, the number of cores, and how they are connected affect the overall performance and energy consumption of the whole system. At the lower level, the neuron model and the crossbar type in each core greatly affect the accuracy of the SNN. These factors must be considered in the early stages of development. Simulating hardware and software mapping helps to forecast the effects of such factors during the development and the actual production of neuromorphic chips. Thus, a simulator is an important tool for the design space exploration of these factors.

When designing a simulator, we need to consider the current trend in increasing the computational power of neuromorphic chips. More chips and systems are being developed with more cores connected. The number of cores has grown from the range of 256-4096 earlier Kapoho Bay, Nahuku systems and IBM TrueNorth chip to 98,304 cores in the current generation of Intel Pohoiki Springs [26]. The size of each core can also increase, depending on the use case. Hence it is imperative that the community can scale the simulation environment to enable simulations or large chips at a reasonable simulation time. Additionally, when considering the space of possible configurations that may need to be tested on many different network architectures, the performance of the simulator can quickly degrade and hinder the design and development process of real neuromorphic chips.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

One particularly important, and costly, task in simulating designs for production chips is trying different RRAM noise models and resistance ranges. These factors affect the accuracy of the SNN models and the cost and energy consumption of the actual chip. It is crucial to efficiently simulate potential configurations to make decisions on which material and design are suitable for the real chip. With the current multi-threaded simulation in [13], each simulation workload requires hours of computation. A typical chip production workflow requires trying thousands of configurations to determine the right design. These configurations include crossbar dimension, non-ideal effects of each potential RRAM material, chip dimension, traffic routing algorithm, NoC model on different SNN models, and test datasets. Speeding up a full system simulator at this stage will expedite the whole process significantly.

III. Related Work

There are many proposals to simulate and implement spiking neural networks, as well as neuromorphic computing systems in general [8], [9], [10], [11], [27], [28], [29], [30]. These range from software functional models that only evaluate the accuracy of the model to detailed hardware and system levels.

For software simulation of SNN, previous studies developed support for describing and running SNN models on popular deep learning frameworks [9], [10], [11], [27]. These studies leverage the convenience and popularity of deep learning frameworks and developed their own solutions as extensions to such frameworks. Nemo [12] used discrete event simulation to scale up in large systems and to simulate very large SNN models. However, no consideration was given to the physical implementation (e.g., different type of neurons, RRAM crossbar noise models). Furthermore, there are no performance results from real-life applications with realistic chip designs other than those of their abstract model being simulated as a proof-of-concept. Kolasa et al. [31] also proposed a framework that targets self-organizing neural networks, a completely different model from SNN. We were unable to compare with this tool because it is a functional exploration tool that gives no performance numbers.

For hardware simulations, there are several works mostly focused on the single core or single crossbar simulation [7]. Some other simulators focus only on the design of digital crossbars and do not provide a detailed simulation of the analog RRAM crossbar [28], [29]. MNSIM [8] is a single-threaded simulator that reports detailed hardware simulation that surpassed SPICE simulation. The main simulated component is a computing core with a crossbar configuration, NoC models like the ideal network and mesh network in Simeuro are not supported. Simeuro differs from this category as we provide both hardware-level data as well as higher level data such as NoC statistics, chip configurations and core allocation information based on realistic workloads.

In system-level simulations, existing works can simulate the hardware extensively to provide performance insights and determine the bottleneck factors of each hardware model. From the simulation results, newer and better hardware models can be proposed. Khalil et al. [32] proposed a simulator targeting a novel "block-based neural network" model. However, it was tested only on very small network models, with no evidence of further scalability. The main goal of their simulator is to improve the implementation flow for FPGAs. Lee et al. [13] developed an extensive simulator at the system level. Their simulator can run multi-threaded using OpenMP on multiple CPU cores. However, when simulating larger models and hardware, the CPU-only OpenMP implementation has scalability limitations.

System-level simulations are important tools in SNN chip design. Scaling has been a serious challenge for existing simulators because fine-grained cross-bar designs and system-level traffic simulations are computationally very demanding. In this paper, we propose Simeuro, a system-level simulator for neuromorphic chips that addresses the scalability issue. Simeuro can run fine-grained low-level simulations of neuromorphic chips with up to 20,000 neurosynaptic cores by using multiple GPUs to accelerate the simulations of DNN inferencing on very large datasets.

IV. A Hybrid CPU-GPU Simulator

A. Overview

Simeuro consists of two main parts: the neurosynaptic core simulator and the network on a chip (NoC) simulator.

The neurosynaptic core simulator views the neuromorphic chip as comprising of a series of neurosynaptic cores with N axons as inputs, M digital leaky integrate-and-fire neurons, and a digital or RRAM crossbar with a size of $N \times M$. A digital crossbar stores the learned weights on its array as signed integers with configurable bit widths for computation, while a RRAM crossbar stores them as pairs of low- and high-resistance states.

The neurosynaptic core simulation is executed on GPUs since it is more suitable for an embarrassingly parallel architecture with a large number of lightweight cores performing the large volumes of calculations needed.

The neurosynaptic cores are connected on a NoC. The NoC simulator uses a 2D-Mesh topology, a popular choice for NoCs, for the communication network that enables spike transmission between routers. In our use case, the NoC transmits spikes through routers. Such an operation is memory intensive and can lead to severe thread divergence when executed on the Single Instruction Multiple Threads (SIMT) execution model of modern GPUs. With this performance consideration in mind, we have opted for a hybrid approach in Simeuro where the neurosynaptic core simulator runs on GPUs while the NoC simulator runs on CPU. The main challenge in the design of Simeuro is to manage the overhead involved in communicating and synchronizing these two simulators.

Fig. 4 shows a high-level overview of Simeuro. The neurosynaptic core simulation takes spike sources as inputs to cores in the input layer and sinks spikes from cores in the last layer for further classification. The simulator can also calculate the energy consumption in the neurosynaptic cores. It is also responsible for simulating different crossbar types, neuron types, and the noise model of real devices.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

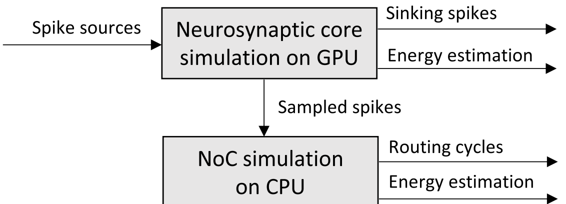


Fig. 4. Overview of Simeuro, a hybrid CPU-GPU parallel simulation platform.

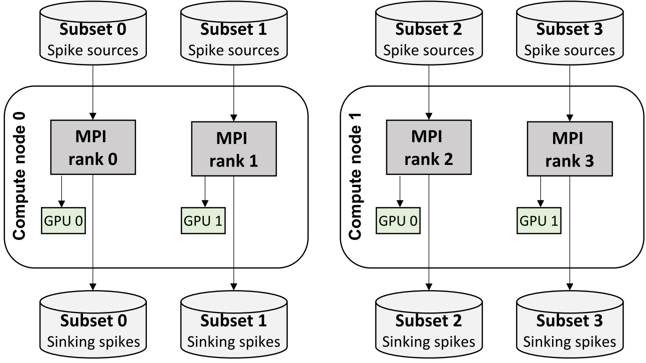


Fig. 5. Distributed computing in the neuromorphic-chip simulation with four instances on two computing nodes.

For the communication network, the NoC simulation produces two metrics to estimate traffic performance in 2D mesh-based network: average routing cycles during which spikes are forwarded from source cores to destination cores, and energy estimation in the NoC. At each simulation cycle, the spikes will be sampled to be passed to the NoC simulation. More detailed information on the sampling technique in the simulator will be introduced in a later section.

Since our goal is to achieve high-detail simulation results with many configurable parameters for research and development, we adopted the configuration model and input/output format from Lee et al. [13]. Our simulator supports the same configurations, basic functionalities, and RRAM noise models as in the aforementioned framework. With all configurable parameters and outputs remaining the same, we only focus on describing the internal design and implementation of our framework in this paper.

Simeuro can take advantage of multi-GPU systems to accelerate core simulations by splitting the input work load into multiple ‘chunks’. For example, to collect simulation statistics of a dataset consisting N images, we can use M GPUs to process N images. In an ideal case, this speedup the simulation time by a factor of M.

In Simeuro we create multiple program instances through the MPI library [33] to perform the simulation tasks simultaneously, where each instance takes a chunk of the dataset and executes the simulation on the GPUs available in the compute node. Fig. 5 shows an example of using two compute nodes, where each compute node hosts two MPI, ranks to execute the simulation. Rank 0 orchestrates the execution by splitting the dataset equally into four subsets and distributing the subsets among other ranks that will perform the simulation simultaneously.

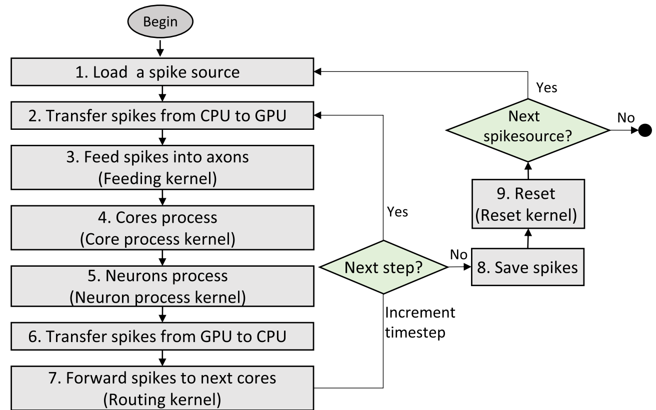


Fig. 6. Workflow of the core simulation in the running state where different steps are executed by different GPU kernels (functions).

V. Neurosynaptic Core Simulation

A. Overview

The neurosynaptic core comprises of a small batch of input axons, corresponding output neurons, and a crossbar. Depending on the type of crossbar and neuron, the accuracy of a SNN and the energy consumption in neurosynaptic cores need to be fully simulated in the core simulation. The heart of a neurosynaptic core is the crossbar that establishes junctions between neurons and axons. The crossbar memory can be configured to set different trained SNN models on the chip. Each component can be described as follows:

- Axons correspond to the rows in the crossbar. They use spikes as inputs to connect neurons through synapses in the crossbar.
- Crossbars memorize the trained weights at the intersection of columns and rows for synapses. The simulator supports two types of crossbars: digital crossbar and RRAM crossbar.
- Neurons fire spikes based on the leaky integrate-and-fire (LIF) model and have configured connections to subsequent axons in other cores.

B. Simulation Workflow

The core simulation has three states during execution: initializing, running, and terminating. In the initializing state, the simulator will create a set of cores. It then loads the synapses and neuron settings for each core from the configuration files. After initialization, the neurosynaptic core simulation enters the running state, and in this state, the simulator will perform the neural computation on the chip. Finally, in the terminating state, the simulator calculates the average energy consumption of the inference per image in the neurosynaptic core simulation. Fig. 6 illustrates the workflow of the neurosynaptic core simulation in the running state on the GPU. The core simulation is a doubly nested loop. The inner loop will take a spike source as input and advance the simulation state using CUDA kernels at every time step, while the outer loop will save output spikes from the simulator to files and load the next spike source from a file as

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

input to the inner loop. In each iteration of the outer loop, the simulation state inside the inner loop will be reset. The core simulation workflow can be summarized in the following steps:

1) Load the next spike source. In this step, the simulator reads a spike source from a local file, which includes a sequence of input spikes for each time step. The inner loop will feed input spikes to axons at each time step iteratively.
2) The input spikes for the current time step are copied from host (CPU) memory to device (GPU) memory.
3) Feeding kernel: input spikes are fed to specific axons located at the cores of the input layer.
4) Core process kernel: the core process integrates synaptic inputs from all axons and updates the membrane voltages in neurons. A CUDA thread updates the state of a neuron and a large number of threads update all the neurons in parallel.
5) Neuron process kernel: the neuron process fires spikes. Neurons will be activated to fire a spike when the membrane voltage subtracting leaky value exceeds the configured threshold.
6) Sinking spikes from neurosynaptic cores in the final layer are copied from GPU to CPU and are temporarily buffered in the host memory.
7) Routing kernel: incoming spikes are forwarded to destination cores using the ideal network model. Note that the ideal network model is always included in the GPU Core simulation. The more complex and realistic NoC model will be simulated separately on the CPU side when requested (description in Section VI). A large number of threads are launched to simultaneously forward incoming spikes from source neurons to destination axons. If the configured maximum number of iterations is reached, the simulator will go to the next step. Otherwise, it increments time steps and continues to step 2.
8) Save output spikes from all iterations to a file to calculate the classification accuracy of SNNs.
9) Reset kernel: reset the simulator states (membrane voltages in neurons, presynaptic spikes in axons, postsynaptic spikes in neurons, etc.) before processing the next spike source. If there are remaining spike sources, the simulator will continue to step 1. Otherwise, the simulator will terminate.

C. Mapping the Core Simulation to GPU's Hierarchy

The core simulation, in either the digital crossbar or the RRAM crossbar, will run on GPUs in a similar fashion. The ideal network model will also run on GPUs to reduce the memory transfer needed between the host and the device. The allocation of which workload to run on the GPUs depends on the user's choice in the configuration file. The core simulation consists of accumulating the synapse value and checking the condition of whether a spike is fired. After the core simulation, the sampled spikes will be fed to the NoC simulator.

Consider the neuromorphic chips to be simulated having C cores, each core has N neurons, A axons, and $S = A \times N$ synapses. We can map each core to CUDA blocks.[2] Because each core typically has its own input, the inputs can be loaded into the shared memory of each block to reduce latency. For example, if a neurosynaptic core has 256 neurons and 256 axons and a neuromorphic chip includes 1,000 cores, the core simulator will launch 1,000 blocks with 256 threads on the GPUs. If a warp contains 32 threads in GPUs, 1,000 blocks with 256 threads will be divided into 8,000 warps. Inside each block, we use each thread to compute a neuron output. The task of each thread is to loop over all axons and accumulate the product of the axons and synaptic weight to a single value. Here, depending on the crossbar model, there might be additional computations (e.g., the RRAM crossbar will have a certain noise model to affect the accumulating value).

After the loop, the accumulated value will be checked against a firing model to produce spike output if the threshold is reached.

With this mapping, the shared memory is used whenever each thread has enough workload to perform. Our mapping will launch C blocks with N threads each. The shared memory overhead is A elements of floating point type. However, the drawback of this mapping is that the number of simulated cores is small. In that case, the number of blocks launched cannot fully utilize the GPU.

D. Memory Mapping and CUDA Optimizations

To optimize the performance of the simulator, we need to consider the access pattern and the usage of memory for the simulation data.

For synaptic values of an entire chip, normally the size of them is very large: we store a 2D array of synapses in the crossbar to a 1D array row-wise in global memory. A neuron in a CUDA thread accesses synapses from global memory and integrates the input spikes multiplied by the synaptic values in the crossbar. Digital crossbars use a single weight to implement one synapse, while RRAM crossbars use a pair of resistances. Also, the RRAM model needs more memory to simulate non-ideal effects. Thus, the memory usage of RRAM is much higher than that of a digital crossbar. This will be discussed in Section VII-C2.

The core simulator also needs to allocate regions in the global memory to store input spikes to axons and output spikes from neurons; usually, their sizes are small because the size of input spikes depends on the length dimension of the axons, while the size of output spikes depends on the length dimension of the neurons in the entire chip. For input spikes, we can take advantage of shared memory to improve computational performance. Each active CUDA block loads the required input spikes from global memory to shared memory before executing the core processing code. The shared memory is accessible by the threads in the same thread block, hence the same simulated core. As discussed above, the shared memory required is a function of the number of input spikes. Due to the limited shared memory for each CUDA block, we only use it to buffer input spikes.

[2]For a general overview on CUDA, we refer readers to the CUDA Programming Guide [34].

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

The core simulator also needs to allocate an area in the global memory to store the settings for each neuron, including a threshold, a membrane voltage, a reset value, connection to the next axons, etc.; its size depends on the number of neurons in the entire chip.

We applied several optimizations to the simulator including the use of shared memory and row-wise coalesced global memory accesses. Our experiments show that using shared memory can improve the core simulation by up to 23%, depending on workload. Likewise, we conducted an experiment to measure the different speeds between access patterns. A naive implementation would use column-wise order to flatten the synapse memory. Compared to column-wise accesses, row-wise accesses can increase the overall simulation speed by up to 7.2×.

VI. Network-on-Chip Simulation

A. Overview of Network-on-Chip Simulation

The *network-on-the-chip* (NoC) provides on-chip communication for neuromorphic chips. The simulator estimates traffic flow and energy consumption during communication using a popular NoC model (introduced by [35]) for 2D mesh-based NoCs. The NoC comprises routers, network interfaces, links, network topology, and flits:

- Links are physical wires to connect adjacent routers and transfer flits between them.
- Flits are atomic units exchanged in the communication network within one routing cycle. Usually, a packet is split into a variable number of flits, including a header flit and multiple body flits [36]. In this paper, the simulator simulates a neuromorphic chip that is specific for spiking neural networks where the spiking value is always one, so the payload data in the body flits can be ignored, and a packet is converted into a header flit with a target address.
- The network interface is a connection interface between the neurosynaptic core and the router. The network interface provides two-way communication to receive or send data and is responsible for buffering spikes and assembling them into packets. Initially, the network interface collects spikes injected from neurons and encodes the spikes into packets with destination addresses, and then forwards them to the associated router. Then, it receives the packets from the associated router and decodes the packets as input spikes to axons for core processing in the next cycle.
- The network topology determines how routers, cores, and links are connected. The simulator implements the 2D-Mesh topology for the communication network, which is a common and simple network topology. This network includes X rows and Y columns. The routers are deployed at each junction point and connected to adjacent routers through links.
- Routers are the most important elements in the communication network and are connected by links and forward incoming flits along routing paths to either the next routers or destination cores. A router consists of five input ports, a crossbar switch, an arbiter, and five output links. The input port has to route logic to determine the next node of a flit.

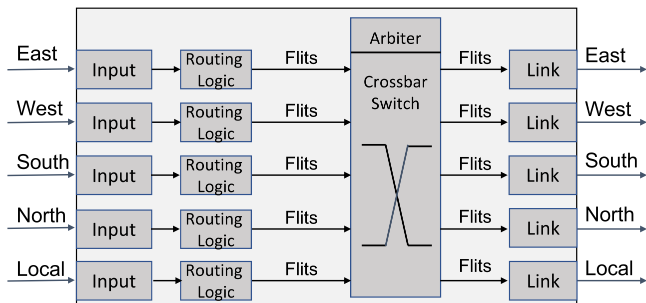


Fig. 7. A generic router architecture used in NoC simulation.

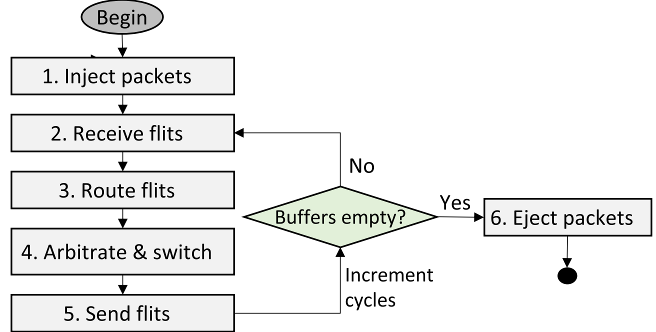


Fig. 8. Flowchart of NoC Simulation in the running state using a multi-core CPU.

The routing logic implements an XY routing algorithm, which moves a flit in the X direction and then moves it in the Y direction. An input port also has a FIFO buffer, which can hold a list of flits. The size of the FIFO buffer can be configured in the simulator. The arbiter is a logical element that controls the order of access to the shared links, the scheduling scheme in the arbiter uses fixed priority arbitration [37]. Fig. 7 illustrates a typical router model.

B. NoC Simulation Workflow

The NoC simulation comprises the same three states as the neurosynaptic core simulation: initialization, running, and termination. In the initialization state, the simulator creates a set of routers in the 2D-Mesh topology, where each router is connected to four adjacent routers through links except for edge routers. A router is associated with a neurosynaptic core. The dimensions (X and Y) of the 2D-Mesh network are specified in the configuration file. After initialization, the NoC simulation enters the running state, in which the simulator transmits incoming packets to the target cores through routers. In the final state, the simulator calculates two metrics: average routing cycles and average energy consumption for spikes' transmission in the NoC simulation. It measures the traffic performance for a specific SNN model in a neuromorphic-computing chip.

Fig. 8 illustrates the NoC simulation workflow in the running state. In this state, the simulator transmits packets until all buffers in the routers are empty. Six steps are repeated in this state. The processing of Step 1 and Step 6 occurs between a network interface and a router. The processing of Steps 2 to 4 occurs

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

within routers, and the processing of Step 5 occurs between routers. The NoC simulation workflow in the running state can be summarized into the following steps:

1) Inject packets: the network interface packetizes the spikes fired from the neurons with destination addresses and injects the packets to the associated router.
2) Receive flits: routers receive flits from network interfaces. Flits are stored in the input ports of a router before being forwarded to the next routers.
3) Route flits: the XY routing algorithm determines the routes for the top flits in the input buffers. Then, flits will request a specific outgoing link after routing.
4) Arbitrate and switch: fixed priority arbitration is used in arbitration. According to routing requests from all input ports, the accesses of shared links are allocated in order of Fixed priority. After arbitration, a crossbar switch is used to establish connections between the input ports and the outgoing links.
5) Send flits: flits on the head of input ports are sent to the next nodes through links after the input ports are successfully guaranteed to access them. If all buffers are empty, then the simulator goes to the next step. Otherwise, it increments routing cycles and goes back to step 2.
6) Eject packets: flits are converted to packets when they arrive at the destination cores and are then ejected from the router to the network interface.

All routers perform the same operations from Step 2 to Step 5 until all input ports are empty.

Algorithm 1: Conventional Parallel Routing With OpenMP Critical Sections.

```
   Input: Spikes from neuron cores: Input_packets
   Output: Spikes to neuron cores: Output_packets
   Output: Routing cycles: Cycles
1  Buff ← Input_packets
2  Cycles ← 0
3  while Buff.length > 0 do
4      #omp_parallel_for
5      for all routers do
6          #omp_critical
7              pending_flits ←
                 routing(Buff, routing_policy)
8              for link ∈ [S, N, W, E, L] do
9                  link.rounted_flit ←
                     arbitration_policy(pending_flits)
10             end
11         end
12         #omp_critical
13             for link ∈ [S, N, W, E] do
14                 Buff ← link.rounted_flit
15             end
16         end
17         local_flits ←
             local_flits ∪ link.rounted_flit;   link == Local
18     end
19     end
20     Cycles ← Cycles + 1
21 end
22 Output_packets ← local_flits
```

C. Parallel NoC Simulation on Multicore CPUs

In Simeuro the NoC simulation uses multi-threaded programming on a multi-core processor to speed up calculations. Processing of all routers is executed in parallel. We use OpenMP to construct parallel regions to parallelize the processing of routers on a multi-core processor. The router consists of the 4-step process described above. Steps 2 to 4 are internal operations and are independent among routers, but step 5 is an inter-router operation in which flits cross two routers through the links. If only one parallel region is constructed by OpenMP to parallelize the processing of routers, as shown in Algorithm 1, a race condition can occur since the routers' processing involves modifying their input buffers from two different threads. A critical section (lock) is a common technique to prevent race conditions from occurring. With the critical section, only one thread can perform the modification of the input buffer in the router at a time. But the disadvantages of critical section lock are obvious: (1) add overhead for each access of input buffers; (2) only one thread can enter the critical section while other threads are blocked and waiting for the resources; (3) deadlock may occur when two routers from different threads are holding and waiting to access the resources from each other. In our NoC simulator, to avoid the race condition, two parallel regions are constructed by OpenMP instead of one region as shown in Algorithm 2. An implicit barrier is created automatically at the end of each parallel region by OpenMP to synchronize all threads. The first parallel region groups the processing of steps 2 to 4 occurring within routers, while the second parallel region groups the processing of step 5 occurring between routers.

D. Optimizing the NoC Simulator

1) Asynchronous CPU-GPU Implementation: In simulating a complex NoC model, the NoC simulator on the GPU consists of repeatedly copying data to the arbitrary destination location (each destination is dynamically determined by the routing algorithm and runtime data) and checking many `if-else` conditions in each router; there is no complex computation involved. This makes the NoC simulator on GPU very slow due to the random global memory access pattern and thread divergence. It is also very difficult to take advantage of memory coalescing when each time the data package is only a spike to a destination. We did implement a GPU version of the NoC simulation, but only to find that it occupies $\approx 99.88\%$ of the total simulation time and is significantly slower than the CPU NoC simulator. In other words, all kernels in Table I only contribute $\approx 0.12\%$ to the total runtime if the NoC is also simulated in the GPU. Results from the Nvidia profiler also indicated that the NoC simulator is bounded by memory-latency with limited optimization opportunities. In contrast, a multicore CPU with its deeper and larger memory hierarchy is better suited for this task.

Thus, we introduce a new implementation that uses both CPUs and GPUs for simulation. Specifically, the core simulation will run on GPUs where it is more effective, while NoC simulation

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: Lock-Free Parallel Routing in Simeuro.

```
    Input: Spikes from neuron cores: Input_packets
    Output: Spikes to neuron cores: Output_packets
    Output: Routing cycles: Cycles
 1  Buff ← Input_packets
 2  Cycles ← 0
 3  while Buff.length > 0 do
 4      #omp_parallel_for
 5      for all routers do
 6          pending_flits ←
              routing(Buff, routing_policy)
 7          for link ∈ [S, N, W, E, L] do
 8              link.rounted_flit ←
                  abitration_policy(pending_flits)
 9          end
10      end
11      end
12      #omp_parallel_for
13      for all routers do
14          for link ∈ [S, N, W, E] do
15              Buff ← link.rounted_flit
16          end
17          local_flits ←
              local_flits ∪ link.rounted_flit;   link == Local
18      end
19      end
20      Cycles ← Cycles + 1
21  end
22  Output_packets ← local_flits
```

TABLE I
BREAK DOWN OF THE EXECUTION TIME OF KERNELS AND MEMORY TRANSFER ON THE GPU BETWEEN TWO CROSSBAR TYPES. HTOD INDICATES HOST TO DEVICE AND DTOH STANDS FOR DEVICE TO HOST MEMORY COPIES, RESPECTIVELY

Kernel Name	Digital Crossbar		RRAM Crossbar	
	Time (ms)	Time (%)	Time (ms)	Time (%)
Core process kernel	549.69	39.6%	2153.18	66.8%
Routing kernel	332.15	24.1%	329.60	10.2%
Neuron process kernel	207.98	15.1%	212.64	6.6%
HtoD MemCopy	154.84	11.2%	400.38	12.4%
DtoH MemCopy	117.58	8.5%	109.50	3.4%
Others	20.42	1.5%	18.40	0.6%

with only data reads and writes between different routers is performed on CPUs. The data required for the NoC simulator that runs on the CPU is copied from the GPU at each simulation cycle in an asynchronous manner. The CPUs run mesh-based NoC simulation, and the GPUs perform the ideal network simulation without a routing mechanism to quickly distribute the spikes and advance to the next core simulation cycle.

2) Subsampling the Data in NoC: Because NoC simulation is a performance bottleneck in our simulator, we apply several optimizations to reduce NoC simulation times. There are two different alternatives that can be applied:

- Use a theoretical queueing model to derive routing cycles [38], [39]. Although this method can yield a very high speedup, the error is considerably high in certain data and network models. Thus, a detailed energy model becomes difficult to achieve.
- Use subsampling to reduce NoC workload [40]. By subsampling a certain amount of data (e.g., 10%), the simulator can output the simulation result with certainty and an error bound. This method is intuitively simple to understand and easy to apply to system-level simulators such as ours.

Thus, we use the subsampling method in our simulator as an option for the user to choose in the configuration file. We apply the method in [40] with a configurable amount of subsampled data to be in the NoC simulator. The full evaluation of this is presented in Section VII.

VII. EVALUATION

In this section, we report the performance and accuracy of the simulator when using different hardware configurations with variable numbers of neurosynaptic cores.

A. Experiments Setup

We conduct experiments on GPU-accelerated ABCI supercomputer[3] which contains thousands of A100 GPUs for multi-GPU results. For single GPU results we use a workstation powered with an Intel(R) Xeon(R) W-2295 CPU @ 3.00 GHz containing 18 cores (36 threads) and an NVIDIA GeForce RTX 2080 Ti GPU. We use the CIFAR-10 [42] dataset in the experiments. The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes with 50,000 training images and 10,000 test images. CIFAR-10 is widely used for benchmarking the performance of neural networks in the field of machine learning. We use a pre-trained neural network model for CIFAR-10 image classification (described in [20]). This model is a binary network, except for the first full-precision transduction layer, which consumes 100 neuromorphic cores as a subnet after the hardware mapping. The spike sources for the neuromorphic-computing chip are generated from the first transduction layer, which converts digital values in CIFAR-10 images to spikes with a single time step. For scaling measurement, we scale up our experiments by duplicating this subnet (100 cores) and spike sources N times, then we can measure a large-chip performance with up to 20,000 cores after 200 times duplication of the subnet and spike sources.

B. Performance & Scalability

First we measure the runtime performance of core simulation on a single Nvidia A100 GPU. We tested 1,000 CIFAR-10 images with different number of physical neurosynaptic cores (from 1000 - 20,000 cores). For the core simulation benchmark, we always use the ideal-network model for NoC. Fig. 9 shows the run-time of our simulator when the crossbar configuration is Digital or RRAM Crossbar. As seen in Fig. 9, the runtime of both RRAM-crossbar and digital crossbar simulations increases linearly as the number of neurosynaptic cores to be simulated increases. The run-times are around 1,069 and 725 seconds

[3]ABCI is the world's first large-scale Open AI Computing Infrastructure [41] and ranks 19th in the Top500 list at June 2022.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

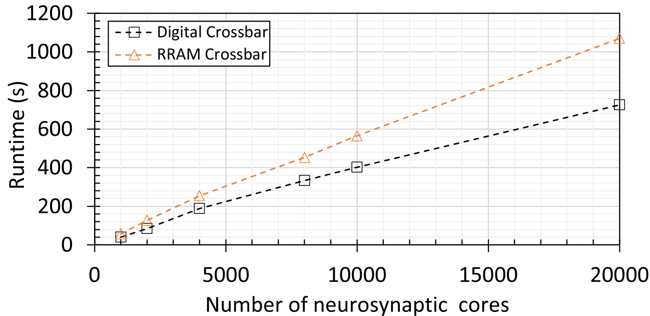


Fig. 9. Simulation runtime for CIFAR-10 images inference by a neuromorphic-computing chip with variable numbers of neurosynaptic cores using a Nvidia A100 GPU between two crossbar types.

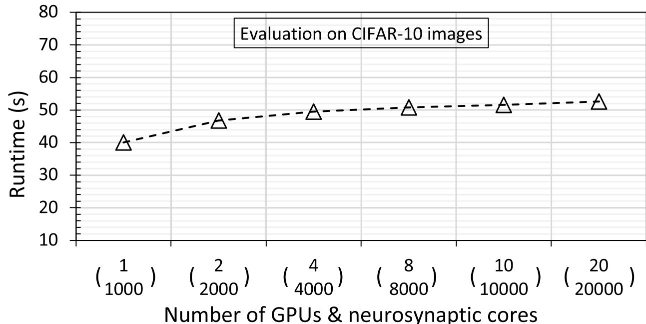


Fig. 11. Weak scaling of the Digital-crossbar simulation for CIFAR-10 images inference, with corresponding changes in both the number of GPUs (top number on the x-axis) and neurosynaptic cores (bottom number of the x-axis).

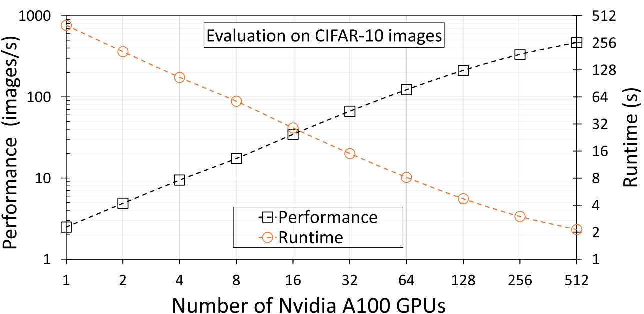


Fig. 10. Strong scaling of the Digital-crossbar simulation for CIFAR-10 images inference while keeping the number of neurosynaptic cores constant 10,000. The performance is plotted on a base-10 logarithmic scale while the runtime is plotted on a base-2 logarithmic scale.

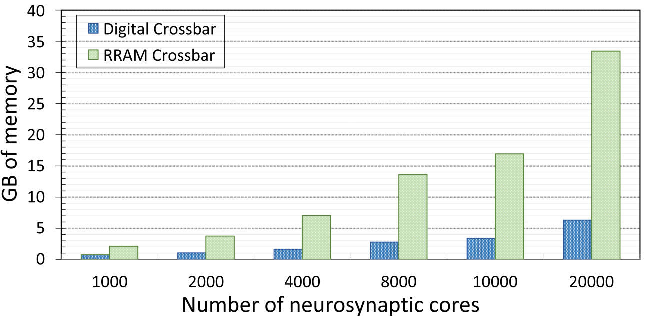


Fig. 12. Memory usage in GPU global memory when different numbers of neurosynaptic cores are simulated.

respectively, when the crossbar configured as RRAM and digital crossbar with 20,000 neurosynaptic cores. In comparison digital Crossbar, RRAM crossbar simulation takes around 40% more time due to more computation required to simulate the RRAM material.

1) Multi-GPU Scaling: We report strong and weak scaling performance on the ABCI supercomputer. This experiment will run Digital-crossbar simulations with a variable number of GPUs or a variable number of neurosynaptic cores. The dataset is be divided equally among the GPUs to be simulated simultaneously. The results of strong scaling measurement are shown in Fig. 10. The strong scaling is measured by running the simulation of 1,000 CIFAR-10 images' inference on a neuromorphic-computing chip with a variable number of GPUs, while keeping the number of neurosynaptic cores constant at 10,000. From Fig. 10 we can see that the run-time for images' classification in the neuromorphic-chip simulator decreases almost linearly with the number of GPUs. In addition, Fig. 10 also shows linear strong scaling of the performance (in images processed in one second) when using different numbers of GPUs. The results of the weak scaling measurement are shown in Fig. 11. The weak scaling is measured by running the simulation of 1,000 CIFAR-10 images' inference on a neuromorphic-computing chip with different numbers of GPUs and with correspondingly scaled number of neurosynaptic cores. From Fig. 11 we can see that as the number of GPUs increases, simulating a large chip with 20,000 cores can achieve similar performance to a small chip with 1,000 cores.

C. Single-GPU Performance

1) Profiling GPU Activities: In order to understand the performance of core simulation on GPUs, we ran the core simulator on the same SNN application on CIFAR-10 images (described in the paper [13]) using 3,680 cores with each crossbar type. We used the `nvprof` profiling tool to collect profiling data about CUDA-related activities on both CPU and GPU. The profiling results are shown in Table I. For both crossbar types, we can see that the core simulator spends most of the time in *Core process kernel* ($\approx 40\%$ in the digital crossbar and $\approx 67\%$ in the RRAM crossbar). Most of the kernels have similar execution times between the two crossbar types. The exceptions are the core simulation in *Core process kernel* and copying simulation data from the host (CPU) memory to device (GPU) memory.

2) Memory Usage: Memory usage depends on the type of crossbar being simulated. Generally, memory on the host (CPU side) has higher capacity than that on the GPU devices. Thus, to determine the upper limit of the size of the network model that our simulator can support, we measured the memory usage of different simulation configurations. Fig. 12 shows the memory usage on GPUs in our simulation with a different number of cores. As we can see, memory usage increases almost linearly with the number of neurosynaptic cores to be simulated, because each core has independent data that needs to be stored. We can also observe that the RRAM crossbar uses significantly more memory than the digital crossbar. The reason is that RRAM crossbars need more data for accurate noise simulation as well as the differential pairs of weights for each output neuron. This

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

contributes to the difference in performance between the two crossbar types on GPU. As shown in Table I, the GPU to CPU memory copy overhead in the RRAM crossbar is $\approx 2.6\times$ that of the digital crossbar.

3) Discussion: In the digital crossbar, the weights are only involved in simple floating point multiply and accumulate operations. For RRAM crossbar, each weight requires an additional two CURAND calls for each weight being read (positive and negative weights in the differential pair in Fig. 3(a)). Processing each weight also involves reading and writing data related to other non-ideal effects [13]. In the Monte Carlo simulation, each thread's workload also depends on the output of the runtime generated random value. This thread divergence also contributes to lower speedup. In concrete numbers, if we simulate a workload with M synaptic weights (typically millions to billions), the digital crossbar only needs to read M weights from the global memory each time step. In the same configuration, just to simulate one of the defects, the RRAM crossbar simulator reads $2 \times M$ weights. To accurately simulate the noise model, there are $2 \times M$ `curand_uniform` calls in each time step of the simulator. The throughput of CURAND is much less than floating point arithmetic [43], [44]. Hence, it is clear that they contribute significant overhead to the RRAM core simulation. This combines with the simulations of other defects in RRAM results in a slower execution time. As can be seen in Table I, *Core process kernel* in RRAM is almost $4\times$ slower than in the digital crossbar.

Currently, the simulation of a neuromorphic chip relies on a single GPU, allowing for a maximum simulation of up to 20,000 neurosynaptic cores. Fig. 12 illustrates the memory usage of $\sim$34 GB for 20,000 cores. However, when dealing with a larger neuromorphic chip, the limitations of a single GPU become apparent: a single GPU simply does not have enough memory to perform the simulation. To scale the simulation, we will discuss a solution to extend our simulator to simulate a large chip across multiple GPUs in Section VIII-B.

D. NoC Simulator Speedup

In the previous section, we reported the performance of the core simulator and used an ideal network as the NoC model. Besides the ideal network, our simulator also supports real-world use cases of other types of NoCs. As described in Section IV, we use the hybrid implementation for NoC simulation, where the core simulation on GPU is interleaved with the NoC simulation on multi-core CPUs. However, through experiments, we found NoC simulation will be very slow when running on the GPU. For example, we conducted three experiments to run NoC simulation on GPUs with different sizes of the 2D-Mesh network: 5×5, 10×10 and 64×64. When the size of the 2D-Mesh network is 5×5, the execution times of NoC simulation on CPUs and GPUs are similar. But when the size of the 2D-Mesh network is 10×10, the execution time of NoC simulation on GPUs is about 5 times slower than the time on CPUs. When the size of the 2D-Mesh network is 64×64, the execution time of NoC simulation on GPUs is about 30 times slower than the time on CPUs. These

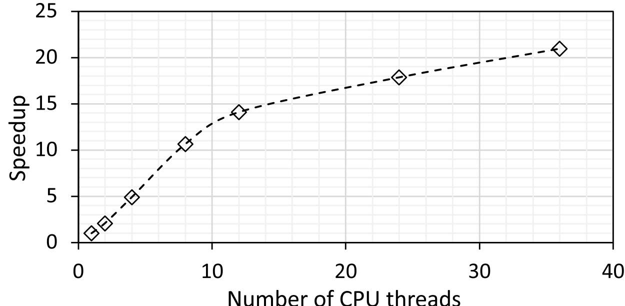


Fig. 13. Scaling a MeshNetwork simulation on CPU with a variable number of threads. The baseline is the simulation time on a single thread.

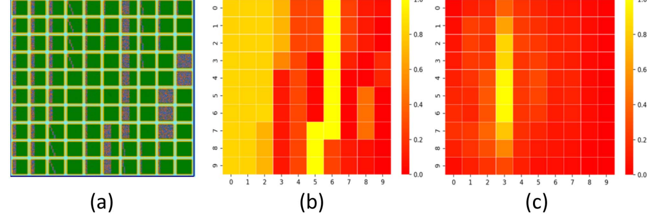


Fig. 14. Energy estimation results during image inference on CIFAR-10 test set by a neuromorphic computing chip with 100 neurosynaptic cores. (a) Synaptic weights on neurosynaptic cores. (b) Energy heatmap of neurosynaptic cores. (c) Energy heatmap of routers in 2D mesh-based NoC.

experiments demonstrate that the NoC simulation is not suitable to run on GPUs.

While NoC simulation exhibits a good scaling performance on Intel(R) Xeon(R) W-2295 CPU @ 3.00 GHz containing 18 cores (36 threads), Fig. 13 shows the actual speedup in NoC simulation with a size of 45×45 2D-Mesh network when scaling the number of threads. When all 18 cores in the CPU are used up, the execution time with 36 threads can be up to $21\times$ faster than with a single thread.

E. Energy Estimation

A micro-level energy model is implemented in the simulator to estimate the energy consumption on the entire neuromorphic chip. The simulator implements all operations in the neurosynaptic cores and NoC, so the micro-level energy model based on these operations is also implemented. More details on Simeuro's energy estimation for neuromorphic chips is available in [45]. We also extend the single node implementation described in [45] to Simuero (i.e., adapt the energy model to multi-nodes), this extension simply merges the results from each node and averages them, then calculates the average energy consumption of the chip. Fig. 14 shows an example of the energy consumption produced by the simulator. Fig. 14. (a) shows the distribution of synaptic weights on the neurosynaptic cores after mapping from a trained SNN model. In this figure, a chip is configured to include 100 neurosynaptic cores with a size of 256×256, each core has a set of junction points, each of which has an x-coordinate, a y-coordinate, and a weight associated with it. The weights from a trained SNN model are quantized as 8-bit signed integers, the quantized weights with zero values, positive values or negative values are mapped to green color, red color

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

TABLE II
THE VARIANCE IN SIMULATION RESULTS AND THE SPEEDUP OBTAINED BY USING SUBSAMPLING METHOD IN OUR NOC SIMULATOR. NO DIFFERENCE IN THE ROUTING CYCLES AND ACCURACY WAS FOUND IN ALL THE TESTS

% data in NoC	Average energy consumption (% difference)	Speedup
Test set A, 1000 images, Average routing cycles = 154, Accuracy=74.1%, Average energy consumption = 1.0076E-7 Joule/image		
5%	-0.055	18.52
10%	0.050	9.16
20%	-0.051	4.84
30%	0.009	3.28
60%	-0.020	1.67
80%	0.005	1.27
Test set B, 5000 images, Average routing cycles = 154, Accuracy=73.9%, Average energy consumption = 1.0077E-7 Joule/image		
5%	-0.030	14.60
10%	-0.006	7.65
20%	0.001	5.47
30%	0.002	3.75
60%	0.008	1.90
80%	0.008	1.43

or blue color respectively. Fig. 14. (b) shows the average energy consumption of inference per image on different neurosynaptic cores. Fig. 14. (c) shows the average energy consumption of inference per image on different routers in the NoC.

F. Sampling Method

We use two test sets to evaluate the proposed sampling method. The result is shown in Table II. The number of images in test set A is 1,000 and the number of images in test B is 5,000. A trained SNN model is mapped onto the chip containing 100 neurosynaptic cores with a size of 256×256. The test images are from CIFAR-10. For comparison, the baseline does not involve any sampling and all data (100%) will be used in the NoC. The sampling method will be tested against this baseline. As can be seen in Table II, our sampling method performs well: the two metrics of average routing cycles and average energy consumption of inference per image in the NoC are identical or similar when we randomly select a small dataset to enter the NoC simulation according to a given percentage. The 'speedup' column shows how much acceleration can be achieved in the average execution time of inference per image on the simulator. We can see the entire simulation has been significantly improved when a small percentage of images are selected in the NoC simulation. Interestingly, we found that the average routing cycles and the accuracy of the SNN model do not change when the percentage of subsampled data is reduced. Accuracy remains the same because the data is actually transferred using an ideal network on GPU.

G. Simulation Verification

Simeuro was also tested for correctness and accuracy. Our simulator is a part of a comprehensive toolchain for estimating the accuracy of SNNs mapped onto neuromorphic hardware. In our toolchain, a user will first convert trained artificial neural networks (ANNs) to SNNs [46], [47], map the models to chips according to their hardware structure [14], [48], [49], quantize the learned weights into n-bit integers, and program them into the crossbars [50]. These processes are complex and repetitive. If the mapping results of the learned models are directly applied to chips, programming errors may occur, and it is not easy to locate and fix them in the hardware. Simeuro allows for designs to be iterated upon quickly before final deployment on real hardware [51], [52]. We verified our simulator by comparing the results between Simeuro and a real neuromorphic chip named as Novena. As shown in Fig. 15, it is in-house designed and its low-level circuit design is reported in [52]. In the verification of accuracy, we used a SNN model for MNIST, which is mapped to 7 neurosynaptic cores with a size of 256×256. Axons of the chip worked with spike trains with a time window of 25 as their input. Spike trains were converted from 10,000 images in the MNIST test set. The learned weights were quantized to 4-bit integers. Simeuro reported an accuracy of 88.71%, which matched the accuracy obtained on the Novena chip. In addition to accuracy, we also verified the energy consumption estimated by our simulator using Novena. We compared the energy consumption estimated by our simulation with actual measurements done on the Novena chip. This demonstrates that our simulator can accurately estimate on-chip energy consumption. In particular, our simulator estimated that the energy consumed per spike on the Novena chip is about 1 picojoule (pJ), which is consistent with the actual hardware measurement. More details were reported in [45].

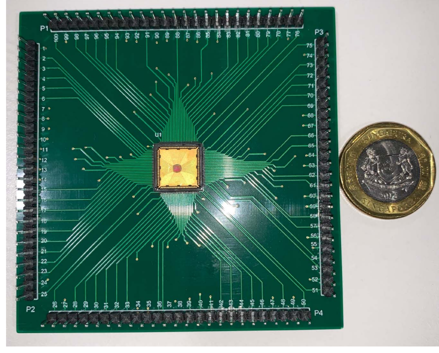


Fig. 15. A fabricated neuromorphic chip named as Novena [52] was designed by Singapore A*STAR's Institute of Microelectronics in 2021. The chip was fabricated with 40 nm technology. A Singapore dollar coin is placed beside it for size comparison.

VIII. CONCLUSION AND FUTURE WORK

A. Conclusion

In this paper, we present Simeuro, a system-level simulator for neuromorphic chips. The simulator uses both GPUs and CPUs to optimize for different parts of the simulation. Our simulator supports different hardware types (digital and RRAM crossbar) with configurable properties and 2D Mesh-based NoC to produce fine-grained statistics on various parts of the system. With the optimized hybrid design, we can execute the core simulation on GPUs (CUDA) and the NoC simulation with multi-threaded programming on CPUs. The size of the models

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

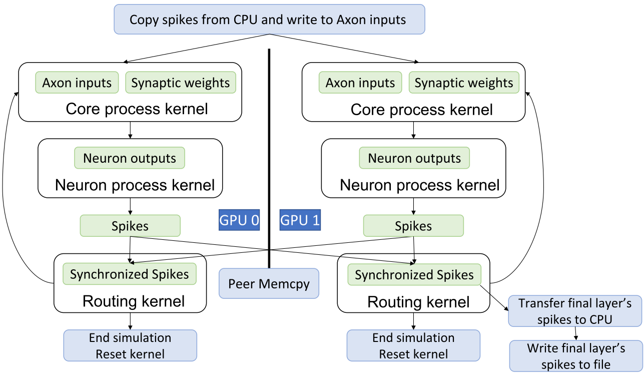


Fig. 16. Prototype of 2-GPU simulation of N neurosynaptic cores. An equal number ($N/2$) of neurosynaptic cores are allocated to each GPU. Two GPUs need to communicate to synchronize spikes data.

that our simulator can handle is the largest reported to date for its class. As technology scales and models grow in size, we believe that our simulator can help design and develop brain-inspired computing methods shortly.

B. Future Work

As future work, we plan to extend our simulator to support large chip simulations with multiple GPUs. Currently, one MPI rank in the simulation only works on a single GPU, limiting a simulated chip to a maximum of 20,000 neurosynaptic cores. However, when simulating larger neuromorphic chips with more neurosynaptic cores the memory requirements exceed the capacity of a single GPU. To address this limitation, we can extend our simulator to support large chip simulations with multiple GPUs by automatically allocating an equal number of neurosynaptic cores to each GPU for simulation. For example, we can simulate N neurosynaptic cores on K GPUs by having one MPI rank splitting them into N/K cores for each GPU to simulate. In Fig. 6, *Core process kernel* and *Neuron process kernel* can be executed independently on each GPU. In *Routing kernel*, because we need to send the spikes to their destination cores, which can be located on any GPU, device-to-device communication is required to synchronize the spikes. Thus, for K GPUs, this solution requires $K \times (K-1)$ peer-to-peer memory transfers to merge and synchronize spikes data among all GPUs before the execution of the *Routing kernel* can begin. While this multi-GPU solution may introduce memory transfer and synchronization overhead, it will enable large chip simulations with more than 20,000+ cores. Fig. 16 provides a visualization of our initial exploration in a 2-GPU system.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1106–1114. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a-68c45b-Paper.pdf

[2] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder–decoder approaches,” in *Proc. 8th Workshop Syntax Semantics Struct. Statist. Transl.*, Doha, Qatar: Association for Computational Linguistics, 2014, pp. 103–111. [Online]. Available: https://aclanthology.org/W14--4012

[3] Y. Deldjoo, M. Elahi, P. Cremonesi, F. Garzotto, P. Piazzolla, and M. Quadrana, “Content-based video recommendation system based on stylistic visual features,” *J. Data Semantics*, vol. 5, pp. 1–15, Jun. 2016.

[4] W. Maass, “Networks of spiking neurons: The third generation of neural network models,” *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608097000117

[5] 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.

[6] P. A. Merolla et al., “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[7] T. Patrick Xiao, Christopher H. Bennett, Ben Feinberg, Matthew J. Marinella, and Sapan Agarwal, “CrossSim: accuracy simulation of analog in-memory computing,” 2017. [Online]. Available: https://github.com/sandialabs/cross-sim

[8] L. Xia et al., “MNSIM: Simulation platform for memristor-based neuromorphic computing system,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1009–1022, May 2018.

[9] A. K. Fidjeland and M. P. Shanahan, “Accelerated simulation of spiking neural networks using GPUs,” in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–8.

[10] D. Yudanov, M. Shaaban, R. Melton, and L. Reznik, “GPU-based simulation of spiking neural networks with real-time performance & high accuracy,” in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–8.

[11] M. Beyeler, K. D. Carlson, T.-S. Chou, N. Dutt, and J. L. Krichmar, “CARLsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks,” in *Proc. Int. Joint Conf. Neural Netw.*, 2015, pp. 1–8.

[12] M. Plagge, C. D. Carothers, E. Gonsiorowski, and N. Mcglohon, “NeMo: A massively parallel discrete-event simulation model for neuromorphic architectures,” *ACM Trans. Model. Comput. Simul.*, vol. 28, no. 4, pp. 1–25, 2018.

[13] M. K. F. Lee et al., “A system-level simulator for RRAM-based neuromorphic computing chips,” *ACM Trans. Archit. Code Optim.*, vol. 15, no. 4, Jan. 2019, Art. no. 64. [Online]. Available: https://doi.org/10.1145/3291054

[14] L. Yang et al., “Coreset: Hierarchical neuromorphic computing supporting large-scale neural networks with improved resource efficiency,” *Neurocomputing*, vol. 474, pp. 128–140, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221018555

[15] M. Tarkov, “Mapping weight matrix of a neural network’s layer onto memristor crossbar,” *Opt. Memory Neural Netw.*, vol. 24, pp. 109–115, Jul. 2015.

[16] T. M. Taha, R. Hasan, C. Yakopcic, and M. R. McLean, “Exploring the design space of specialized multicore neural processors,” in *Proc. Int. Joint Conf. Neural Netw.*, 2013, pp. 1–8.

[17] A. Ankit, A. Sengupta, P. Panda, and K. Roy, “RESPARC: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks,” in *Proc. 54th Annu. Des. Automat. Conf.*, 2017, pp. 1–6.

[18] A. Shafiee et al., “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[19] M. Hu et al., “Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication,” in *Proc. 53rd ACM/EDAC/IEEE Des. Automat. Conf.*, 2016, pp. 1–6.

[20] T. Luo et al., “NC-net: Efficient neuromorphic computing using aggregated sub-nets on a crossbar-based architecture with non-volatile memory,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 9, pp. 2957–2969, Sep. 2022.

[21] S. Ambrogio, S. Balatti, A. Cubeta, A. Calderoni, N. Ramaswamy, and D. Ielmini, “Statistical fluctuations in HfOx resistive-switching memory: Part II–Random telegraph noise,” *IEEE Trans. Electron Devices*, vol. 61, no. 8, pp. 2920–2927, Aug. 2014.

[22] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, “Training and operation of an integrated neuromorphic network based on metal-oxide memristors,” *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[23] S. Han, J. Lee, and K. Choi, “Tree-mesh heterogeneous topology for low-latency NoC,” in *Proc. Int. Workshop Netw. Chip Architectures*, 2014, pp. 19–24.

[24] A. Tavakkol, R. Moraveji, and H. Sarbazi-Azad, “Mesh connected crossbars: A novel NoC topology with scalable communication bandwidth,” in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl.*, 2008, pp. 319–326.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

[25] L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, and R. Locatelli, "NoC topologies exploration based on mapping and simulation models," in *Proc. 10th Euromicro Conf. Digit. Syst. Des. Architectures Methods Tools*, 2007, pp. 543–546.
[26] Y. S. Yang and Y. Kim, "Recent trend of neuromorphic computing hardware: Intel's neuromorphic system perspective," in *Proc. Int. SoC Des. Conf.*, 2020, pp. 218–219.
[27] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "SpykeTorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Front. Neurosci.*, vol. 13, pp. 625, 2019, doi: 10.3389/fnins.2019.00625.
[28] R. Preissl et al., "Compass: A scalable simulator for an architecture for cognitive computing," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.
[29] K. Minkovich, C. M. Thibeault, M. J. O'Brien, A. Nogin, Y. Cho, and N. Srinivasa, "HRLSim: A high performance spiking neural network simulator for GPGPU clusters," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 2, pp. 316–331, Feb. 2014.
[30] T. Luo et al., "An FPGA-based hardware emulator for neuromorphic chip with RRAM," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 2, pp. 438–450, Feb. 2020.
[31] M. Kolasa and R. Długosz, "An advanced software model for optimization of self-organizing neural networks oriented on implementation in hardware," in *Proc. 22nd Int. Conf. Mixed Des. Integr. Circuits Syst.*, 2015, pp. 266–271.
[32] M. Khalil-Hani, V. P. Nambiar, and M. N. Marsono, "Co-simulation methodology for improved design and verification of hardware neural networks," in *Proc. IEEE 39th Annu. Conf. Ind. Electron. Soc.*, 2013, pp. 2226–2231.
[33] MPI, "Open source high performance computing," Accessed: May 30, 2022. [Online]. Available: https://www.open-mpi.org/
[34] Nvidia, "Cuda programming guide," Accessed: Oct. 30, 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[35] W.-C. Tsai, Y.-C. Lan, Y.-H. Hu, and S.-J. Chen, "Networks on chips: Structure and design methodologies," *J. Elect. Comput. Eng.*, vol. 2012, 2012, Art. no. 2.
[36] J. Flich, *Flow Control*. Boston, MA, USA: Springer, 2011, pp. 683–689. [Online]. Available: https://doi.org/10.1007/978--0-387-09766-4_316
[37] S. Ansari and G. Surumi, "A modified NoC router architecture with fixed priority arbiter," *Int. J. Sci. Res.*, vol. 4, no. 10, pp. 923–928, 2015.
[38] M. Lai, L. Gao, N. Xiao, and Z. Wang, "An accurate and efficient performance analysis approach based on queuing model for network on chip," in *Proc. Int. Conf. Comput.-Aided Des.*, 2009, pp. 563–570.
[39] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "A comprehensive and accurate latency model for network-on-chip performance analysis," in *Proc. 19th Asia South Pacific Des. Autom. Conf.*, 2014, pp. 323–328.
[40] W. Dai and N. E. Jerger, "Sampling-based approaches to accelerate network-on-chip simulation," in *Proc. IEEE/ACM 8th Int. Symp. Netw.-on-Chip*, 2014, pp. 41–48.
[41] ABCI, "ABCI supercomputer," Accessed: Jun. 30, 2022. [Online]. Available: https://abci.ai/
[42] kriz, "Cifar10 dataset," Accessed: May 30, 2021. [Online]. Available: https://www.cs.toronto.edu/%7Ekriz/cifar.html
[43] NVIDIA, "Random number generation on NVIDIA GPUs," Accessed: May 30, 2021. [Online]. Available: https://developer.nvidia.com/curand
[44] NVIDIA, "NVIDIA a100 datasheet," Accessed: May 30, 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf
[45] Z. Wang et al., "NCPower: Power modelling for NVM-based neuromorphic chip," 2020. [Online]. Available: https://doi.org/10.1145/3407197.3407619
[46] Y.-L. Chen, C.-C. Lu, K.-C. Juang, and K.-T. Tang, "Conversion of artificial neural network to spiking neural network for hardware implementation," in *Proc. IEEE Int. Conf. Consum. Electron. - Taiwan*, 2019, pp. 1–2.
[47] N.-D. Ho and I.-J. Chang, "TCL: An ANN-to-SNN conversion with trainable clipping layers," in *Proc. 58th ACM/IEEE Des. Autom. Conf.*, 2021, pp. 793–798.
[48] Y. Ji et al., "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints," in *Proc. IEEE/ACM 49th Annu. Int. Symp. Microarchitecture*, 2016, pp. 1–13.
[49] C. Zou, X. Cui, Y. Kuang, and X. Wang, "Mapping convolutional neural networks onto neuromorphic chip for spike-based computation," in *Proc. China Semicond. Technol. Int. Conf.*, 2021, pp. 1–3.
[50] J. Yang et al., "Quantization networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7300–7308.
[51] V. P. Nambiar et al., "0.5v 4.8 pj/sop 0.93uw leakage/core neuromorphic processor with asynchronous NoC and reconfigurable lif neuron," in *Proc. IEEE Asian Solid-State Circuits Conf.*, 2020, pp. 1–4.
[52] M. M. Wong et al., "A 2.1 pj/sop 40 nm SNN accelerator featuring on-chip transfer learning using delta STDP," in *Proc. IEEE 51st Eur. Solid-State Device Res. Conf.*, 2021, pp. 95–98.

Huaipeng Zhang received the master's degree in computer engineering from the Shenyang University of Technology, in 2003, and the master's degree in knowledge engineering from the National University of Singapore, in 2019. He is currently a senior research engineer with the Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), 1 Fusionopolis Way, #16-16 Connexis, Singapore 138632, Republic of Singapore. His areas of Interests include high-performance computing, simulation, machine learning.

Nhut-Minh Ho received the PhD degree from the National University of Singapore, in 2020. He is currently a postdoctoral researcher with the Department of Computer Science, National University of Singapore. His research interests include approximate computing and GPU code optimization.

Dogukan Yigit Polat received the bachelor's degree in computer engineering from Bilkent University, Turkey, in 2018. He is currently working toward the PhD degree with the National University of Singapore since 2019, and he worked as a software engineer for a year after his graduation. His research lies in lossless conversion from conventional Artificial Neural Networks (ANNs) to Spiking Neural Networks (SNNs), training noise resistant ANNs to be deployed on SNN hardware with noisy Resistive Memory (ReRAM) systems and Neural Architecture Search (NAS) for network models to be deployed on highly constrained hardware platforms such as SNN chips.

Peng Chen received the BE degree in navigation from Dalian Maritime University, China, in 2005, the ME degree in traffic information engineering and control from Shanghai Maritime University, China, in 2007, and the PhD degree from the Tokyo Institute of Technology, Japan, in 2020. He is a researcher with the National Institute of Advanced Industrial Science and Technology (AIST). Also, he is working as a visiting scientist with RIKEN Center for Computational Science (RIKEN-CCS), Japan. His research interests include parallel computing, image processing, and machine learning.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.

Mohamed Wahib received the PhD degree in computer science from Hokkaido University, Japan, in 2012. He is currently a team leader of the "High Performance Artificial Intelligence Systems Research Team" with RIKEN Center for Computational Science (R-CCS), Kobe, Japan. Prior to that he worked as is a senior scientist with AIST/TokyoTech Open Innovation Laboratory, Tokyo, Japan. His research interests revolve around the central topic of high-performance programming systems, in the context of HPC and AI. He is actively working on several projects including high-level frameworks for programming traditional scientific applications, as well as high-performance AI.

Truong Thao Nguyen received the BE and ME degrees from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2011 and 2014, respectively, and the PhD degree in informatics from the Graduate University for Advanced Studies, Japan, in 2018. He is currently working with Digital Architecture Research Center, National Institute of Advanced Industrial Science and Technology (AIST), where he focuses on the topics of High Performance Computing system, Distributed Deep Learning and beyond.

Jintao Meng received the BS and MS degrees in computer science from Central China Normal University, Wuhan, in 2005 and 2008 respectively, and the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2016. He is an associate researcher with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interests include high performance computing, bioinformatics, and graph computing.

Rick Siow Mong Goh received the PhD degree in electrical and computer engineering from the National University of Singapore. He is the director of the Computing & Intelligence (CI) Department, Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore, where he leads a team of more than 80 scientists in performing world-leading scientific research, developing technology to commercialization, and engaging and collaborating with industry. His current research interests include artificial intelligence, high-performance computing, block chain, and federated learning.

Satoshi Matsuoka received the PhD degree from the University of Tokyo, in 1993. He had been a full professor with the Global Scientific Information and Computing Center (GSIC), The Tokyo Institute of Technology since 2001, and the director of the joint AIST-Tokyo Tech. Real World Big Data Computing Open Innovation Laboratory (RWBC-OIL) since 2017, and will become a Specially Appointed professor with Tokyo Tech starting 2018 along with his directorship with R-CCS. He has been the leader of the TSUBAME series of supercomputers that have won many accolades such as world no. 1 in power-efficient computing. He also leads various major supercomputing research projects in areas such as parallel algorithms and programming, resilience, green computing, and convergence of big data/AI with HPC. He has been a major driving force behind the development of the next-generation flagship supercomputer of Japan, the supercomputer Fugaku. In June 2020 Fugaku won the first place in four major rankings of supercomputer performance, Top500, HPCG, HPL-AI, and Graph500.

Tao Luo received the bachelor's degree from the Harbin Institute of Technology, Harbin, China, in 2010, the master's degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, and the PhD degree from the School of Computer Science and Engineering, Nanyang Technological University, Singapore, in 2018. He is currently a senior research scientist and group leader with the Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore. His current research interests include high-performance computing, quantum computing, reconfigurable computing system, hardware.software co-exploration, efficient artificial intelligence, and its application.

Weng-Fai Wong (Senior Member, IEEE) received the BSc degree from the National University of Singapore, in 1988, and the DrEngSc degree from the University of Tsukuba, Japan, in 1993. He is currently an associate professor with the Department of Computer Science, National University of Singapore. His research interests include computer architecture, compilers, and high-performance computing.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on July 05,2024 at 02:27:44 UTC from IEEE Xplore. Restrictions apply.