# Performance Evaluation of Spintronic-Based Spiking Neural Networks using Parallel Discrete-Event Simulation

ELKIN CRUZ-CAMACHO, Computer Science, Rensselaer Polytechnic Institute, Troy, United States
SIYUAN QIAN, University of Illinois at Urbana-Champaign, Urbana, United States
ANKIT SHUKLA, University of Illinois at Urbana-Champaign, Urbana, United States
NEIL MCGLOHON*, Computer Science, Rensselaer Polytechnic Institute, Troy, United States
SHALOO RAKHEJA, University of Illinois at Urbana-Champaign, Urbana, United States
CHRISTOPHER CAROTHERS, Computer Science, Rensselaer Polytechnic Institute, Troy, United States

Spintronics devices that use the spin of electrons as the information state variable have the potential to emulate neuro-synaptic dynamics and can be realized within a compact form-factor, while operating at ultra-low energy-delay point. In this paper, we benchmark the performance of a spintronics hardware platform designed for handling neuromorphic tasks.

To explore the benefits of spintronics-based hardware on realistic neuromorphic workloads, we developed a Parallel Discrete-Event Simulation model called Doryta, which is further integrated with a materials-to-systems benchmarking framework. The benchmarking framework allows us to obtain quantitative metrics on the throughput and energy of spintronics-based neuromorphic computing and compare these against standard CMOS-based approaches. Although spintronics hardware offers significant energy and latency advantages, we find that for larger neuromorphic circuits, the performance is limited by the interconnection networks rather than the spintronics-based neurons and synapses. This limitation can be overcome by architectural changes to the network.

Through Doryta we are also able to show the power of neuromorphic computing by simulating Conway's Game of Life (GoL), thus showing that it is Turing complete. We show that Doryta obtains over 300× speedup using 1,024 CPU cores when tested on a convolutional, sparse, neural architecture. When scaled-up 64 times, to a 200 million neuron model, the simulation ran in 3:42 minutes for a total of 2000 virtual clock steps. The conservative approach of execution was found to be faster in most cases than the optimistic approach, even when a tie-breaking mechanism to guarantee deterministic execution, was deactivated.

CCS Concepts: • **Hardware** → **Integrated circuits**; **Spintronics and magnetic technologies**; **Neural systems**.

Additional Key Words and Phrases: spiking neural networks, spintronic devices, chip performance, energy estimation, parallel discrete event simulation, game of life, turing completeness



*Author now at IBM Research, contribution made while affiliated with Rensselaer Polytechnic Institute.

Authors' Contact Information: Elkin Cruz-Camacho, Computer Science, Rensselaer Polytechnic Institute, Troy, New York, United States; e-mail: cruzce@rpi.edu; Siyuan Qian, University of Illinois at Urbana-Champaign, Urbana, Illinois, United States; e-mail: siyuanq3@illinois.edu; Ankit Shukla, University of Illinois at Urbana-Champaign, Urbana, Illinois, United States; e-mail: ankits4@illinois.edu; Neil McGlohon, Computer Science, Rensselaer Polytechnic Institute, Troy, New York, United States; e-mail: mcglon2@rpi.edu; Shaloo Rakheja, University of Illinois at Urbana-Champaign, Urbana, Illinois, United States; e-mail: rakheja@illinois.edu; Christopher Carothers, Computer Science, Rensselaer Polytechnic Institute, Troy, New York, United States; e-mail: chrisc@cs.rpi.edu.


ACM 1558-1195/2024/7-ART
https://doi.org/10.1145/3649464

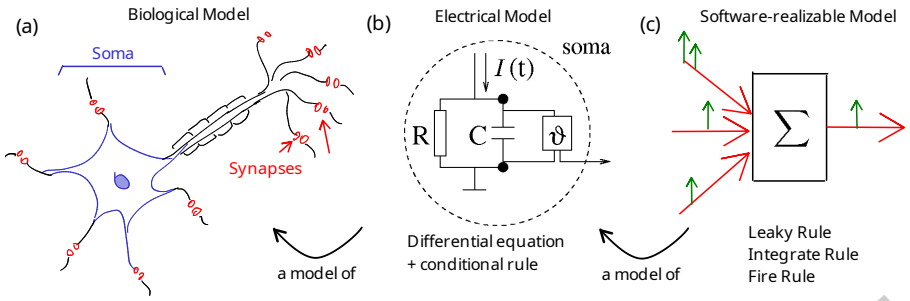



Fig. 1. Source of inspiration for Spiking Neural Networks and abstraction levels. (a) The biological spiking neuron. (b) A model (Leaky-Integrate Fire) of the biological spiking neuron [28, Fig. 4.1]. (c) Software-realizable neuron model: neuron (soma) and synapses.

## 1 Introduction

Neuromorphic computing is a non-von Neumann approach to computing inspired on the brain. It has been proposed as a solution to reduce the high energy consumption needs of machine learning tasks. For example, as part of the DARPA SyNASPE program, IBM created an instance of a spiking neuromorphic processor, called TrueNorth, capable of multi-object detection and classification from real-time video input consuming only 63 milliwats [49]. The chip has 4096 neurosynaptic cores with a total of 1 million spiking neurons and 256 million re-configurable synapses. Another example is Intel's *Loihi*, 2018, a spiking neuromorphic processor capable of performing on-chip learning [20]. Even with these low energy consumption chips in development, there is hope that we have not yet reached the minimum energy limit for neuromorphic computing; according to Hasler and Marr [33], biological neurons and synapses, if realized truly efficiently in silicon, would be able to compute $10^{18}$ multiply-accumulate operations (MAC) per second using only 1 watt of power. To that end, it is important to continue improving the state-of-the-art and further reduce the hardware costs associated with neuromorphic computing.

To seek out improvements at the hardware level, we could make use of spintronic devices for the fabrication of electronic neurons and synapses in a brain-inspired architecture. Spintronics devices, including antiferromagnets and ferromagnets, made using magnetic materials are non-volatile and can mimic the dynamics of biological neurons and synapses in hardware in an energy-efficient and compact form-factor, i.e, occupying around the same space of a single transistor. This energy and area efficiency might open a new artificial intelligence (AI) paradigm endowed with real-time learning, adaptation, and prediction. Such brain-inspired AI hardware can be used in remote, "edge computing" environments with size, weight, and power constraints.

To facilitate our evaluation of spintronics-based neuromorphic hardware, we have developed a multi-scale modeling and simulation approach where physical hardware costs (i.e., energy, area, and latency) are calculated for key neuromorphic operations including neuron integration, neuron fire, and signal communication. We limit our approach to the analysis of these components and leave for future work the estimation of additional components required to manufacture a chip such as transducers and amplifiers. These performance models are then imported into the neuromorphic simulator, Doryta[1].

The key contributions of this work include:

(1) Modeling approach to quantify the performance of spintronic devices for neural networks:

[1]Doryta can be found at https://doi.org/10.5281/zenodo.11585299.

- Quantification of the energy, chip area, and runtime performance of spintronics-based neurons and synapses for neural network inferencing tasks resulting in three to six orders of magnitude improvement in energy-delay product over CMOS designs.
- Exploration of the effect that interconnect size and wire width have on performance metrics.

(2) Development of Doryta, a deterministic, parallel spiking neural network simulation platform able to simulate neuromorphic applications, validated against existing spiking neural network tools.
(3) Evaluation and validation of Doryta:
- Implementation of Conway's Game of Life as a pure neuromorphic application model for Doryta, thus demonstrating that spiking neural networks are Turing-complete.
- Evaluation of parallel Doryta simulation performance wherein the Game of Life neuromorphic application model obtains over 300× speedup using 1024 CPU cores.
- In depth analysis of what makes the conservative approach for synchronization generally faster than optimistic when running discrete-event simulations such as Game of Life, and the impact of different configuration changes such as size of the model and the activation of the tie-breaker mechanism.

## 2 Background

In this section, we briefly introduce the components of spiking neural networks, present the spintronic devices that simulate those components, and summarize parallel discrete-event simulation.

### 2.1 Spiking Neural Networks

Ubiquitous Artificial Neural Networks (ANNs) are not the only relevant bio-inspired development to come out of studying the brain. Spiking Neural Networks (SNNs) [6], as the name implies, are based on the biological spiking neurons. The biological spiking neuron is a prevalent neuron type in the brain. It responds to stimuli in the form of *spikes*. The simplest electrical model of the spiking neuron is the Leaky-Integrate Fire (LIF) Neuron [6], see Figure 1, which can be described by only two procedures: *a differential equation* that determines how the voltage of the neuron changes as a function of time

$$C\frac{dV(t)}{dt} = I(t) - \frac{V(t) - V_e}{R}; \tag{1}$$

and *a conditional rule* that dictates how the neuron discharges

$$\texttt{if } V(t) > V_{\text{th}} \texttt{ then set } V(t) = V_{\text{reset}} \texttt{ and fire} \tag{2}$$

where $V(t)$ is the voltage of the neuron at time $t$, $I(t)$ is the input current to the neuron, $C$ is the capacitance, $R$ is the resistance, and $V_e$, $V_{\text{reset}}$ and $V_{\text{th}}$ are the resting, reset and threshold potentials, accordingly.

Notice that there is no general analytical solution for the two rules, leaving us to seek a numerical solution. To numerically simulate Equation 1, we discretize it into $\Delta t$ time steps

$$V(t + \Delta t) = V(t) + \Delta t\frac{-(V(t) - V_e) + I(t)R}{\tau}, \tag{3}$$

where $\tau \equiv RC$.

The neurons in a neural network are interconnected via axons and synapses. We use the term *synapse* to refer to the connection between two neurons at a logical level: from a neuron that fires and sends a spike to a neuron that receives it. At the hardware level, neurons are connected via physical wires or *interconnects*, while the synapse is a circuit element capable of changing the intensity of a signal on a wire. All synapses have two attributes: how much current arrives at the neuron and a delay. For simplicity, we assume that all synapses have

the same delay, one clock cycle. This implies that $I(t)$, in Equation 3, represents the summation of all currents from the synapses that receive a spike at time $t$, i.e.

$$I(t) = \sum_i W_i S_i(t), \tag{4}$$

where $W_i$ is the weight for the $i$-th synapse, and $S_i(t)$ is either 1 or 0 indicating whether a spike was received or not to the $i$-th synapse at time $t$, respectively.

With neurons and synapses, we can construct many complex structures, also called neural network (NN) architectures. A full SNN model is defined by an NN architecture and three procedures (or operations): the leak operation (Equation 3), the integrate operation (Equation 4), and the fire operation (Equation 2). At the hardware level, we realize these three operations using spintronics-based spiking neurons and synapses, and metallic interconnects.

## 2.2 Spintronics Neurons and Synapses

The field of spintronics is expected to support semiconductor-based microelectronics in 'More-than-Moore' and 'Beyond Moore' information technologies [31, 46]. In contrast to conventional electronics that deal with the charge of an electron, spintronics utilizes the spin of an electron to manipulate, transmit, store and detect information. Spintronic devices can be fabricated using back-end-of-the-line CMOS processes and, therefore, realized in modern fabrication facilities without much re-tooling. At the heart of a spintronic device is a magnetic material: ferromagnetic (FM) or antiferromagnetic (AFM), which acts as the active component and displays neuro-synaptic dynamics when perturbed by external input (e.g., current pulse or magnetic fields). The spin configurations of FM and AFM materials are distinct, which makes them functionally unique as the building blocks of neuromorphic hardware.

FM-based spintronics nano-devices, such as magnetic tunnel junctions, are commonly used for storage, sensing, logic, interconnections, and as non-linear radio-frequency oscillators [35]. Recently, it was experimentally demonstrated that FM-based nonlinear oscillators could be used to build circuits that embed neural functionality and can perform speech and digit recognition with high accuracy [75]. FM devices have also been used for realizing synaptic behavior in hardware [8]. Furthermore, AFMs display current-tunable spiking dynamics, which could be used to replicate the functionality of spiking neurons in an NN architecture.

*2.2.1 Antiferromagnetic Spiking Neurons.* Due to their unique spin arrangement, AFMs possess no net magnetization. When a spin-polarized electric current is injected into an AFM, it can display spiking dynamics, which are detectable in the form of an output voltage signal. Thus, AFM-based signal generators can emulate spiking neurons in a compact form-factor in hardware [41, 67]. Figure 2(a) shows an example of an AFM spiking neuron based on a non-collinear, chiral semi-metal, like $Mn_3Sn$, while the time-domain input and output of the neuron are shown in Figure 2(c). In order to excite spiking dynamics, an input spin current that exceeds a certain threshold is provided to the AFM. Based on the material parameters and the dimensions, the AFM neuron's spike rate and performance can be efficiently tuned [41, 67]. The neuron spike can be detected in the form of a voltage signal via the anomalous Hall effect (AHE)[37, 76] through a structure as the one shown in Figure 2(a) or via tunneling magnetoresistance (TMR) [22] as shown in the setup of Figure 2(b). The latter has only been theoretically predicted, while AHE has been experimentally demonstrated in a variety of chiral AFMs [62, 74]. Other detection methods like anisotropic magnetoresistance (AMR) [24] and tunneling anisotropic magnetoresistance (TAMR) [77, 81] may also be used in the case of $Mn_3Sn$, although such methods have a weaker output signal compared to TMR and AHE at 300 K. In the case of collinear insulating AFMs, like NiO, a voltage signal may be detected in a non-local spin valve[68] setup or via the inverse spin Hall effect[18] in a bilayer structure of AFM and heavy metal.

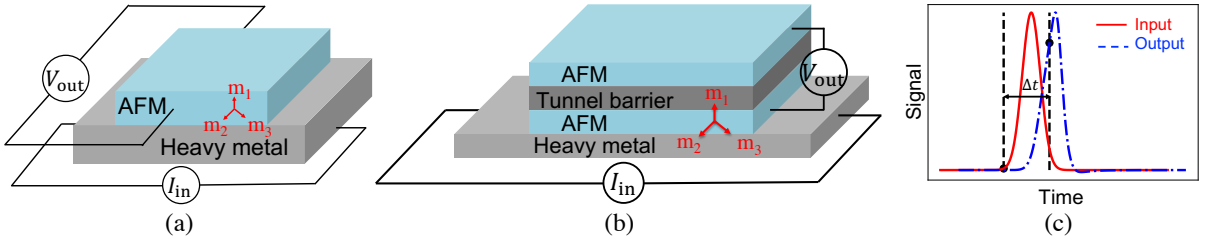



Fig. 2. AFM neuron based on (a) anomalous Hall effect readout and (b) tunneling magnetoresistance readout. (c) A qualitative sketch of the time-domain output signal, $V_{out}$, of the neuron, versus the input signal, $I_{in}$. In (a) and (b), $\mathbf{m_1}$, $\mathbf{m_2}$, and $\mathbf{m_3}$ represent the magnetization vectors of the AFM. In (c), the neuron latency is the timing difference between the input signal arrival and the time the neuron's angular velocity reaches a threshold, which is $10^{10}$ rad/s for $Mn_3Sn$ and $2 \times 10^{12}$ rad/s for NiO.

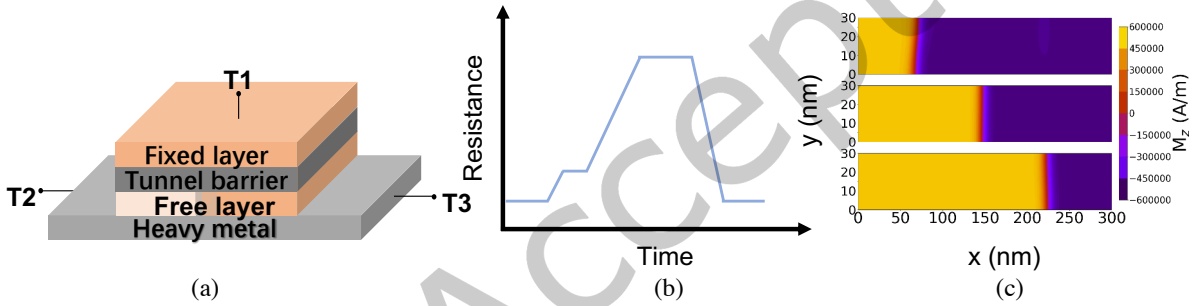



Fig. 3. (a) Schematic of a spintronic synapse where the free layer and the fixed layer are made of ferromagnetic materials. Input current is applied across terminals T2 and T3, while the output voltage is measured across T1 and T3. (b) A representative response of the ferromagnetic synapse over time due to applied input current. (c) A representative result showing the movement of domain wall under applied training current simulated using MuMax3c.



*2.2.2 Ferromagnetic Non-volatile Synapses.* Memristive dynamics based on domain wall (DW) movement can be easily produced by stripe-shaped FM structures, such as Figure 3(a). The device response to input current can be seen in Figure 3(b), which highlights the ability of the device to store real-valued weights with plasticity. Thus, FM materials can act as compact and non-volatile hardware emulators of synapses. The conductance of the device is determined on a training phase. During the training phase, current flows between terminals T2 and T3. The synapse's conductance is set by the magnitude and the duration of $I_{in}$. Figure 3(c) shows the movement of domain wall motion with increasing input pulse durations. During inferencing, the output current between terminals T1 and T3 is measured. The output current is given as the product of the voltage across T1 and T3, and the memristor's conductance. The memristors can be electrically connected in an analog cross-bar fashion such that the net current flowing through the bit line is the weighted sum of the memristors' conductance multiplied by the input voltage.

### 2.3 Parallel Discrete-Event Simulation

Parallel Discrete-Event Simulation (PDES) is an efficient method for modeling the behavior of complex systems with discrete interactions between many entities and is a natural match for the independent, discrete behaviors in spiking neural networks. A PDES simulation is made up of agents or entities known as Logical Processes (LPs) and timestamped events triggered from and to LPs. These LPs are each mapped to the various cores or Processing Elements (PEs) that may exist. In our simulation environment, a single core or MPI process is home to one PE. Given a fixed number of LPs, the more PEs we have, then, the fewer LPs will exist within each PE.

Effective parallelization does not come for free. Computing on a distributed environment requires a well-founded synchronization strategy. Two main synchronization strategies exist for PDES: conservative and optimistic approaches. *Conservative approaches* [7, 16] ensure that events are processed in timestamp order by stalling the simulation until it is "safe" to process each event. *Optimistic approaches* [40] allow events to be processed out of order, but provide a mechanism to detect and erase incorrect event computations. To track time, each LP has their own clock. The Global Virtual Time (GVT) is the minimum clock time across LPs, anything occurred before is considered to "have happened" and any redundant information from that time can be forgotten.

ROSS (Rensselaer Optimistic Simulation System) is a framework for developing parallel discrete-event simulations. ROSS has demonstrated highly scalable, massively parallel event processing capability for both conservative and optimistic synchronization approaches [2, 12]. ROSS' conservative execution is inspired by the YAWNS protocol [52], utilizing an event creation lookahead window restriction that ensures events cannot be created in a way that causes out-of-order processing. ROSS' optimistic execution is accomplished by implementing the Time Warp protocol [38, 39] which works with virtual time [40] for event time management. ROSS mitigates Time Warp state-saving overheads via *reverse computation* [13], where rollback is realized by performing the inverse of the individual operations that were executed in the event computation. When reverse computation is not possible (e.g, floating point operations are not reversible), we make use of incremental state saving for rollback [57].

## 3 Physical Performance Estimation of Spintronic Materials

To estimate the performance of a chip, we estimate first its building blocks (neurons, synapses and interconnects) and add them up in larger units (cores) which in turn make up the chip. In this section, we report the methodology and performance metrics for neuronal spiking, synaptic integration, and data communication via metal interconnects. This subsection is used as a building block for Subsection 5.1 where the performance estimation of specific networks are computed.

### 3.1 Antiferromagnetic Neurons

We consider spiking neurons implemented using two different types of AFM materials: $Mn_3Sn$ (metallic) and NiO (insulating). These AFM neurons generate a spike with an auto-reset when triggered with a current pulse that exceeds a threshold value, which depends on the material parameters and dimensions of the neuron. For a typical 4-nm thick NiO-based neuron, the critical spin current density to excite the neuronal dynamics is typically $1 \times 10^7$ A/cm$^2$ [55]. With a Gaussian input current signal, the latency is related to both the magnitude and the full width at half maximum (FWHM), $t_{\text{halfwidth}}$, of the current density. With $t_{\text{halfwidth}}$ = 5 ps, the latency for NiO-based neuron is 10 ps. For typical material parameters, considering a Spin Hall angle $\theta_{\text{SH}} = 0.05$ at the NiO/Pt interface [36], the critical charge current density to fire a NiO neuron is $4 \times 10^8$ A/cm$^2$. Thus, the energy consumed for a spiking event in a 120 nm × 40 nm NiO-based neuron is around 933 aJ. As for a 4-nm thick $Mn_3Sn$-based neuron, the input spin current density is $2 \times 10^6$ A/cm$^2$ and the latency is approximately 7 ps assuming $t_{\text{halfwidth}}$ = 10 ps. Considering a spin Hall angle $\theta_{\text{SH}} = 0.056$ at the $Mn_3Sn$/Ru interface [50], the critical current density is $3.6 \times 10^8$ A/cm$^2$. Therefore, the energy dissipation of a 120 nm × 40 nm $Mn_3Sn$ neuron is 2.8 aJ per spike. The faster response and lower current density of $Mn_3Sn$ contribute to its energy efficiency compared with that of NiO.

For comparison, a digital CMOS neuron —consisting of two 8-bit registers, an 8-bit adder, 8 NAND gates, 8 inverters, and three 8-state elements— occupies 110 $\mu$m$^2$ area for 15-nm technology node [53]. The energy consumption per spike for the digital CMOS neuron is 136 fJ, while its operating frequency is 1.58 GHz. The analog CMOS neuron is based on an analog-to-digital converter read circuitry composed of 32 CMOS inverter cells and a synapse, which occupies 0.69 $\mu$m$^2$ cross-sectional area [53, 54]. The energy consumption of the analog CMOS neuron is $\approx$ 140 fJ and its operating frequency is $\approx$ 503 MHz.

### 3.2 Ferromagnetic Synapses

In the case of the memristive domain wall-based FM synapses considered here, a $450 \times 30 \times 1$ nm$^3$ device can host up to 16 distinct resistance levels [45, 65]. In order to ensure that the output current is sufficient to fire the neurons, the voltage applied to the synapse must be

$$V_{\text{applied}} = I_{\text{neu}} G_{\text{max}}, \tag{5}$$

where $G_{\text{max}}$ is the maximum conductance of the synapse obtained when the synapse's free layer magnetization is parallel with that of the fixed layer (see Fig. 3 for free and fixed layers). This implies that $V_{\text{applied}}$ is proportional to the neuron's input current. As shown in Table 1, the conductance range of the synapse is from $5 \times 10^{-4}$ $\Omega^{-1}$ to $16.9 \times 10^{-4}$ $\Omega^{-1}$ with CoFe ferromagnet layer [65]. The corresponding read voltage is set to 0.03 V for $Mn_3Sn$ and 0.38 V for NiO based on the threshold current density of neurons in the network. The latencies of the synapses connected with $Mn_3Sn$ and NiO are both 0.13 ps as shown in Table 1. Consistent with the input and output signal of the neurons, we assume that the read voltage signal applied to the synapse is also Gaussian with $t_{halfwidth} = 10$ ps and the corresponding energy dissipation per synaptic event is 7.8 aJ for synapses connected to $Mn_3Sn$ neurons and 983 aJ for synapses connected to NiO neurons.

By comparison, a digital CMOS synapse, consisting of an 8-bit SRAM register and a state element, consumes $\approx$ 170 fJ energy, while its latency is 0.64 ps [53]. The footprint of the digital synapse is estimated as 1.38 $\mu$m$^2$. For an analog CMOS synapse, comprising two operational transconductance amplifiers, the energy consumption is $\approx$ 2 fJ, latency is 19 ps, while its area is 0.17 $\mu$m$^2$ [53, 54].

### 3.3 Interconnects

The aforementioned neurons and synapses are interconnected to each other via Cu/low-$\kappa$ technology. The energy of these interconnect can be evaluated as

$$E_{\text{ic}} = C_l l V^2, \tag{6}$$

where $C_l$ is the per-unit-length interconnect capacitance which is dependent on the wire width, $l$ is the interconnect length, and $V$ is the supply voltage, which for the $Mn_3Sn$ neuron is 3.2 mV while for NiO is 26 mV. We model the

Table 1. Metrics of the DW-based ferromagnetic synapse with $Mn_3Sn$-based and NiO-based neurons.

| **Parameter** | **$Mn_3Sn$** | **NiO** |
|---|---|---|
| Conductance range ($10^{-4}$ $\Omega^{-1}$) | 5-16.9 | 5-16.9 |
| Sensing voltage (Volt) | 0.03 | 0.38 |
| Latency (picoseconds) | 0.13 | 0.13 |
| No. of conductance states | 16 | 16 |
| Energy dissipation (atto-Joules) | 7.8 | 983 |

interconnect length for neurons, $l_{\text{ic,neu}}$, and that for synapses, $l_{\text{ic,syn}}$, according to

$$\begin{aligned} l_{\text{ic,syn}} &= \sqrt{a_{\text{syn}} s_{\text{core}}}, \\ l_{\text{ic,neu}} &= \sqrt{a_{\text{core}} c_{\text{layer}}}, \end{aligned} \tag{7}$$

where $a_{\text{syn}}$ ($a_{\text{core}}$) is the synapse (core) area, $s_{\text{core}}$ is the number of synapses per core, and $c_{\text{layer}}$ is the number of cores per layer.

At the core level, the RC-delay dominates the total delay of the interconnects. Thus, we consider it as part of the synapse delay and evaluate it as

$$\tau_{\text{syn,ic}} = 0.69(R_{\text{ic}} C_{\text{ic}} + R_{\text{load}} C_{\text{ic}} + R_{\text{ic}} C_{\text{load}}), \tag{8}$$

where $R_{\text{ic}} = \rho \frac{L}{A} l_{\text{ic}}$ is the interconnect resistance, $C_{\text{ic}} = C_l l_{\text{ic,syn}}$ is the capacitance of interconnects, $R_{\text{load}}$ is the effective resistance of a synapse, and $C_{\text{load}}$ is the capacitance of the synapse.

At the chip level, the interconnect delay is associated with the neuron operations and is thus evaluated as

$$\tau_{\text{neu,ic}} = \frac{C_l l_{\text{ic,neu}} V_{\text{neu}}}{I_{\text{neu}}}, \tag{9}$$

where $I_{\text{neu}}$ is the input current of the neuron, and $V_{\text{neu}}$ is the neuron voltage. $I_{\text{neu}} = J_{\text{neu}} a_{\text{neu}}$, where $J_{\text{neu}}$ is the neuron's input current density, while $a_{\text{neu,HM}}$ is the neuron's side section area of heavy metal layer. Table 2 lists the values of $J_{\text{neu}}$ and $V_{\text{neu}}$ for both $Mn_3Sn$ and NiO neurons.

Interconnects' resistivity increases non-linearly as the device size shrinks and this causes a worsening performance of the interconnects at smaller dimensions. The leading terms in increasing the resistivity are surface scattering and grain boundary scattering [25]:

$$\rho_{\text{ic}} = \rho_{0,\text{ic}} + \rho_{0,\text{ic}} \lambda_{0,\text{ic}} \frac{3(1-p)}{4d} + \rho_0 \lambda \frac{3R}{2D(1-R)} \tag{10}$$

where $\rho_{0,\text{ic}} = 1.67\ \mu\Omega\cdot$cm is the Cu bulk resistivity, $\lambda = 39.5$ nm is the mean free path of electrons in bulk Cu, $p = 0.5$ is the specularity parameter for surface scatterings, and $R = 0.3$ is the grain boundary reflectivity. Assuming an interconnect aspect ratio (i.e., the ratio of the thickness and width) of 2 and Ta/TaN liner thickness of 3 nm, the interconnect width, $d = w_{\text{wire}} - 6$ nm and the thickness T = 2$w_{\text{wire}}$-6 nm which is also considered as the grain size $D = T$.

The interconnect capacitance per unit length is given as [29]:

$$C_l = \varepsilon_{\text{ox}} \left[ 1.15 \left(\frac{W}{H}\right) + 2.8 \left(\frac{T}{H}\right)^{0.222} \right] + 2\varepsilon_{\text{ox}} \left[ 0.03 \left(\frac{W}{H}\right) + 0.83 \left(\frac{T}{H}\right) - 0.07 \left(\frac{T}{H}\right)^{0.222} \right] \left(\frac{S}{H}\right)^{-1.34} \tag{11}$$

Table 2. Performance metrics of AFM neurons.

| | **Value** | |
|---|---|---|
| **Performance metric** | NiO (insulator) | $Mn_3Sn$ (metal) |
| Input electric current | 0.64 mA | 57 $\mu$A |
| Input voltage | 0.16 V | 5.6 mV |
| Output spike voltage | 0.4 mV | 5.6 mV |
| Latency | 10 ps | 7 ps |
| Energy dissipation | 930 aJ | 2.8 aJ |
| Area | $40 \times 120\ \text{nm}^2$ | $40 \times 120\ \text{nm}^2$ |

where $\varepsilon_{ox} = 2.55\varepsilon_0$ [26] is the permittivity for porous SiCOH, $W$ and $T$ are the width and thickness of interconnects, respectively, while $S$ is the spacing between the interconnects and is assumed equal to $W$. The distance from the ground plane $H = 5$ nm. The typical values for $w_{wire} = 10, 20, 30$ nm are $C_l = 0.31$ pF/m, 0.52 pF/m, 0.76 pF/m respectively.

### 3.4 Chip-level benchmark

Instead of thoroughly designing each part of the chip, we introduce empirical factors to approximate the area associated with peripherals and to accommodate the design rules for component spacing. All cores in our work are connected using a crossbar architecture. The core area is given as

$$a_{\text{core}} = (a_{\text{neu}} n_{\text{core}} F_{\text{neu}} + a_{\text{syn}} n_{\text{in}} n_{\text{out}} F_{\text{syn}}) F_{\text{core}}, \tag{12}$$

where $F_{\text{syn}} = 2$, $F_{\text{neu}} = 2$ and $F_{\text{core}} = 2$ represent the empirical area factor for synapses, neurons and cores, respectively, and $n_{\text{in}}$ ($n_{\text{out}}$) is the number of input (output) neurons for the core.

In case the number of input neurons is smaller than the synapses per neuron, the core area is estimated using the convolution core architecture [53]. Replacing the input neuron numbers by the synapse numbers per neuron, we obtain

$$a_{\text{core}} = (a_{\text{neu}} n_{\text{core}} F_{\text{neu}} + a_{\text{syn}} s_{\text{neu}} n_{\text{out}} F_{\text{syn}}) F_{\text{core}} \tag{13}$$

The fan-in of AFM neurons is considered to be infinite because the input current sums up in the interconnects naturally. All the cores in the same layer operate simultaneously and each core operation delay is determined by one synaptic operation and one neuron operation. The delay per core is given as

$$\tau_{\text{core}} = \tau_{\text{neu}} + \tau_{\text{syn}} + \tau_{\text{neu,ic}} + \tau_{\text{syn,ic}}. \tag{14}$$

The energy consumption per core is the summation of energy dissipated in all the synapses and neurons. The energy consumption of a core is workload-dependent because both the active synapse rate and the neuron fire rate depend on the workload-related parameters presented to the neurons and synapses. The active synapses per neuron and the active neurons per inference (per core) are

$$s_{\text{act,neu}} = s_{\text{neu}} s_{\text{act}}, \tag{15}$$

$$n_{\text{act,core}} = n_{\text{core}} n_{\text{act}}, \tag{16}$$

where $s_{\text{act}}$ and $n_{\text{act}}$ are the ratio of active synapses and neurons per layer, respectively.

Adding all up, the energy consumption per core is

$$E_{\text{core}} = E_{\text{syn}} s_{\text{act,neu}} n_{\text{core}} + E_{\text{neu}} n_{\text{act,core}}. \tag{17}$$

At the chip level, the area and energy consumption are the summation of all cores:

$$a_{\text{chip}} = \sum_i \sum_j a_{\text{core},ij}, \tag{18}$$

$$E_{\text{chip}} = \sum_i \sum_j E_{\text{core},ij}, \tag{19}$$

where $i$ is the layer index, while $j$ is the core index in a specific layer. It is worth noting that all cores of the same layer operate in parallel and thus the chip latency is given as

$$\tau_{\text{chip}} = \sum_i \max_{\text{layer},i}(\tau_{\text{core},j}). \tag{20}$$

## 4 Doryta: Simulating Spiking Neural Networks

Doryta [2] is an architecture-agnostic and deterministic Spiking Neural Network simulator built on ROSS. Doryta's job is not to train a network for a task, i.e., no neuron parameter is altered in the simulation of the network, instead, it is intended to be a reliable, deterministic, and time-aware simulator of SNNs.

Doryta differs from its inspirational predecessor, NeMo [60], in several regards. First, NeMo was written in a combination of C, C++, and Lua, while Doryta is written in pure C. Second, NeMo was originally designed to simulate the TrueNorth [14] chip, and generalizing it to any arbitrary architecture proved a difficult task. In contrast, Doryta is written to be architecture agnostic, and therefore, more flexible, which allows for recursive neuron connections.

We base our SNN implementation on the LIF neuron model for its combination of simplicity and capability. Although there are dozens of neuron models implemented in libraries, many of them, however complex, can be boiled down to behavior similar to that of the LIF neuron, i.e., a neuron can be modeled by the application of three simple rules: the leak rule, the integrate rule and the fire rule. We demonstrate the capabilities and computational power of the LIF neuron model (and its Turing completeness) in Subsection 4.2. More realistic biological neuron models and other complex are out of our scope as we focus on digital neurons.

In the following subsection (4.1), we explain Doryta's implementation details. We then present two outstanding applications for SNNs: in subsection 4.2, Conway's Game of Life, and, in subsection 4.3, image classification with the LeNet architecture.

### 4.1 Implementation Details

Doryta is implemented as a ROSS model divided up into multiple modules: driver LPs (neuron LP), layouts, model-loaders, and neuron types. Here, we explain in detail each module and related concepts such as "Doryta modes", event types, and $\Delta t$-step.

*4.1.1 LPs and Events.* Each LP in Doryta represents a single neuron and is composed of three parts: an ID, a list of synapses (weighted connections to neurons), and a pointer to the LIF neuron parameters (potential, capacitance, current, resting and reset potential, and threshold). New neuron models can be implemented as C `structs` to replace the default LIF neuron model pointer.

In a time-stepped simulation, the state of the system is updated one delta-time step at a time ($\Delta t$-step). This $\Delta t$-step is model dependent and determines the granularity of the simulation. The smaller the $\Delta t$-step, the more precise the simulation, at the cost of a larger computational time. Given the nature of PDES, this $\Delta t$-step in Doryta has to be explicitly encoded as an event. We call this event a *heartbeat event*.

In the LIF neuron model there are three rules: leak, integrate, and fire. Notice that, typically, in a time-stepped simulation all three rules should be executed one after the other on each $\Delta t$-step, but this is not the case in DES. If there is no positive leak, integration will only happen when the neuron is suspected to fire, i.e, when a spike arrives. The leak operation is applied on every $\Delta t$-step. These two concepts can be captured by two events:

- **Heartbeat event**: It applies the rules: leak and fire, in that order. It is scheduled every $\Delta t$-step. It is received, processed, and sent by each neuron to itself. There is at most one heartbeat event per neuron at any given point.
- **Spike event**: It applies the integration rule (aka, summation) to the neuron, i.e., for the LIF neuron, the spike's current is added up to the input current $I(t)$ at time $t$. It can be scheduled at any point in time. Spike events can be created in two ways: loaded when the simulation initializes or created by a neuron.

[2]Doryta is an amalgam composed of the name "Dory" and the Spanish-origin suffix "ita". Dory comes from the movie "Finding Nemo" where a forgetful regal blue tang, called Dory, finds herself accompanying a clownfish in an adventure to find his son, Nemo. Doryta is the spiritual successor to NeMo, another SNN simulator built on ROSS. Doryta, however, is simpler, more modular and flexible — which, we believe, is the essence of Dory the fish.

Although, on biological neurons, spikes are continuous changes in voltage and not instantaneous changes, this instantaneous approximation is acceptable in our context. Our objective is not to reproduce exactly the behavior of biological neurons, but our objective is to reproduce the behavior of typical neuromorphic implementations. Therefore, computational models that are of interest to Doryta align with existing hardware implementations and thus suffer no discrepancies.

We make use of incremental state saving to allow us for the rollback of events under optimistic execution [1]. Heartbeat events are an example of self-initiating models, which have been proven to lead to good performance [51].

Due to the possibility that two or more events could occur with the same timestamp (an event tie), we lean on the tie-breaking feature of ROSS [47] to maintain simulation determinism and validity. This will also guarantee that the order of any two independent, simultaneous events is explicitly randomized. We have implemented *user-defined event priorities* to guarantee that heartbeat events must always be processed before spike events, should they occur at the same timestamp. Our implementation of user-defined event priorities takes advantage of the sorting mechanism imposed by the sequence of tie-breaking values (defined by the tie-breaking mechanism). The sequence is extended with an extra value on front which is user-defined and defaults to 1. This allows the user to have control over the priority of the events if they occur at the same timestamp, yet it has to be used with care as it can easily break the tie-breaking mechanism. The user knows what to do and they can break the determinism of the simulation if they so desire.

*4.1.2 Modes.* When we can guarantee that a neuron does not have positive leak, then the neuron is guaranteed to fire if only if it receives a spike. Thus, without positive leak, there is no reason to schedule a heartbeat event every single $\Delta t$-step, otherwise we would have a recurring heartbeat event consuming resources when no spikes arrive. We can leverage this fact and run simulations in one of two modes:

- **Needy mode:** A heartbeat event is scheduled every $\Delta t$-step regardless of neuron input.
- **Spike-driven mode:** A heartbeat event is scheduled only after a spike is received, preventing the creation of heartbeat events when they are not needed.

Needy mode allows for neurons with positive leak, while spike-driven does not, yet it has a speedup advantage.

To make simulations on both modes semantically the same, two constraints should always be maintained: heartbeats are allowed to be scheduled only at specific timestamps, and a new operation (big-leak) must yield (approximately) the same as computing the operation $n$ times.

To exemplify these two constraints, let us find out when should a heartbeat event be scheduled for a $\Delta t$-step of 1.0. In needy mode, the first heartbeat would be scheduled at time 0.0, and once it is processed, it would schedule a heartbeat at time 1.0, then at 2.0, and so on. In spike-driven mode, no heartbeat would be scheduled at time 0.0. Instead, if a spike arrives at time $t$ then we would have to update the state of the neuron to that point in time (applying the big-leak operation) and then we would schedule a heartbeat for time $\lfloor t \rfloor + 1$. If updating the state (big-leak operation) behaves the same as applying the leak operation consecutively, then we can ensure that with a $\Delta t$-step of 1.0 both simulations will produce the same firing pattern. That being said, for other $\Delta t$-step values like 0.1, although entirely reasonable, one cannot guarantee the same firing pattern due to a loss in precision of floating-point arithmetic operations (addition is *not* repeated multiplication for some values like 0.1 when represented in binary). Thus a careful choice of floating-point $\Delta t$-steps is crucial. Both operation modes are exemplified in Figure 4.

*4.1.3 Delta-time step and Leak operation.* In needy mode, the next heartbeat event is scheduled a $\Delta t$-step from whenever the previous heartbeat event is processed. In spike-driven mode, heartbeat events are not scheduled by heartbeat events but by spike events. When a spike arrives, it runs the integration operation, and then checks

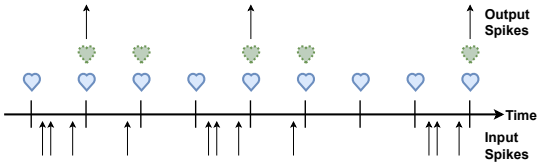



Fig. 4. Heartbeat scheduling for needy (solid blue hearts) compared to spike-driven (dotted green hearts) modes.

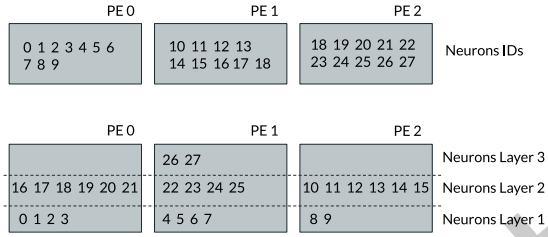



Fig. 5. Mapping configurations. Top: A single fully connected network with 28 neurons. Bottom: 3-layer SNN with dimensions [5 10 16 2] (input 5, output 2).

whether a heartbeat event has already been scheduled and not yet processed (a $O(1)$ operation). If there is no heartbeat event scheduled, it will schedule one for the time:

$$t_{\text{heartbeat}} = \left( \left\lfloor \frac{t_{\text{spike}}}{\Delta t} \right\rfloor + 1 \right) \times \Delta t \tag{21}$$

Subsequent spikes arriving before $t_{\text{heartbeat}}$ will not trigger the scheduling of more events. Given the nature of floating-point number arithmetic, we recommend using $\Delta t$-steps equal to a power of 2 as that is more resistant to floating-point precision errors.

The leak operation (see Equation 3) is a discretization of the differential equation for the neuron behaviour (see Equation 1). As noted before, when no spikes arrive at the neuron, there is no need to update the state of the neuron. It is possible to determine the state of the neuron if the input current ($I(t)$) is zero (or constant) over any period of time, which means that we can compute the new state at once instead of needlessly applying the leak rule many times. This new computation is called the big-leak operation and it is equal to:

$$V(t) = V_e + e^{\frac{-t}{\tau}} \times (V_i - V_e) \tag{22}$$

where $t$ is the time elapsed since the state of the neuron was last computed.

*4.1.4 LP Mapping and Layouts.* Neurons are assigned in groups to each PE. In Doryta, a batch of neurons is reserved to a range of PEs. There can be as many batches as necessary for any simulation. Figure 5 shows two different mappings using the same number of neurons.

On top of the neuron mapping, we have the connections between neurons, synapses. We have implemented two kinds of connections: 'all-to-all' connections and 2D convolutional connections. An all-to-all connection is defined from a range of output neurons to a range of receiving neurons. A fully connected layer is an example of an 'all-to-all' connection (see an example in Section 4.3). A 2D convolutional connection is also defined between a range of output neurons and receiving neurons, with some additional parameters (input image width, kernel parameters (width and height), padding and striding). The total number of synapses for the convolutional connection is at most $K_w \times K_h \times N_r$ where $N_r$ is the total number of receiving neurons, and $K_w$ and $K_h$ are the

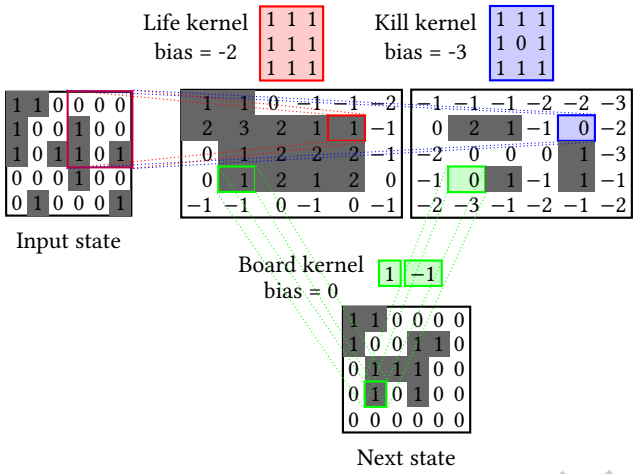



Fig. 6. Conway's Game of Life as a 2-layer convolutional Neural Network. The first layer has a padding of 1 and a kernel with two filters (Life Kernel and Kill Kernel). The second layer has no padding and a kernel of $1 \times 1$, two input channels and one filter (the Board Kernel). A neuron fires (in grey) when its summation (plus bias) surpasses 0, i.e., the activation function for both layers is the step function centered on 0.5.

width and height of the kernel. Note that the number of connections/synapses for a convolutional connection are much smaller than for an all-to-all connection.

All-to-all and convolutional connections are pervasive in ANN applications because of their simplicity and geometric characteristics. In Section 4.2, we show how to construct a grid for Conway's Game of Life made out of four convolutional connections.

## 4.2 Neuromorphic application: Conway's Game of Life and Turing Completeness

Conway's Game of Life [27] (GoL) is an extensively studied cellular automata, a fascinating mathematical construction built out of a grid of cells and a list of rules that determine how the state of the cells change in time. Each cell in the grid is in one of two states: dead or alive, and each cell has eight neighbours. If the grid has a border, then some cells will have less than 8 neighbours. The rules of GoL are simple: a cell stays alive if there are two (2) or three (3) neighbouring cells alive, otherwise it dies; and, a cell comes to life if it has exactly three (3) alive neighbours, otherwise it stays dead.

As noted by [69], it is possible to encode the rules of GoL, and thus simulate GoL, as a multi-layered Neural Network. In Figure 6, we show a 2-layer NN which simulates one step of GoL using as activation function the step function centered on 0.5. Notice that the shape of the network's output (the third layer) is the same as its input. This allows us to connect the output of the last layer as input to the first layer, thus building a recurrent network.

The first convolution is composed of a kernel of size $3 \times 3$ and two filters. We can analyze this kernel by breaking it up into two pieces, one per filter. The first $3 \times 3$ kernel corresponds to the Life kernel, which when applied counts the number of cells alive on groups of 9 cells at the time. The bias for the Life kernel ($-2$) is added up to these counts resulting in the Life number. It can be seen in Figure 6 that the Life number for the uppermost-rightmost quadrant in the input state is equal to 1 (in red). Note that all numbers are natural numbers. If the Life number is bigger than zero, then either the neuron was dead and at least three neighbours are alive or it is alive and at least two neighbors are alive; in either case, the neuron output should be 1 (the neuron fires and a spike is sent). The second $3 \times 3$ kernel corresponds to the Kill kernel, which when applied counts the number of

cells alive around a cell. The bias for the Kill kernel (−3) is added up to this count resulting in the Kill number. If the Kill number is bigger than zero, this means that the number of cells alive around a central cell is bigger than 4, which is not a good outcome for the central cell as it cannot be alive with that number of neighbors regardless of its current state (dead or alive).

In Figure 6, the Kill number for the cell centered in the uppermost-rightmost quadrant is 0 (in blue). The two "images" resulting from the application of the first convolution (Life and Kill images) are binary, containing either 1s and 0s or spikes and no-spikes, and they tell us which cells have a chance to be alive, and which must be stopped and killed (respectively). The second convolution (between the second and third layers) is in charge of determining the "life" status of each cell given the output from the Life and Kill images. The Board kernel distinguishes the input from the Life image or Kill image. All values in the Life image are multiplied by 1 and all values in the Kill kernel are multiplied by −1. The two resulting values, per cell, are added up, giving us the Board number. If the Board number is zero, then the cell is dead, but if it is one, then it is alive. In Figure 6, the green box on the next state image shows the result of adding up $1 \cdot 1$ and $0 \cdot -1$.

A cell in GoL using the 2-layer NN strategy from above can be represented by three neurons, one for the Board kernel and two for the Life and Kill kernels. The simulation takes two clock cycles ($\Delta t$-steps) to simulate one time step of GoL. In the first clock cycle, the input state is sent as spikes from the first to the second layer following the Life and Kill kernel weights and the neurons fire if the conditions apply. In the second clock cycle, the neurons that fired on the previous cycle send a spike to the first layer where it is determined whether the cell is alive or dead.

The neuron parameters for GoL are: $V_e$, $V_{reset}$ of 0, $C$ of 0.5, $R$ of 1 and threshold equal to $0.5 - bias_i$, where $bias_i$ is the bias for the neuron $i$ in one of the three possible neuron groups: Board, Life or Kill. The synapses' weights are: 0 if the neuron being connected corresponds to the same cell and the kernel is Kill, −1 if the synapse connects from the group of Kill neurons to the Board neurons, and 1 in any other case. See Table 3(a) for GoL's crossbar configuration for a grid of size $20 \times 20$. The number of input lines for the Board layer is three times that of the size of the grid because it collects the result of the Life and Kill layers as well as the initial state input.

A natural corollary from GoL as a SNN is that: SNNs are Turing-complete! GoL has been shown to be Turing-complete [4]. Thus, we have shown that the requirements for SNN Turing-completeness are at most: two neuron parameters (threshold and leak), one synaptic parameter (weight) and recurrent connections. Furthermore, it is possible to modify the 2-layer convolutional NN to make use of only non-negative numbers, which makes it possible to set a fixed threshold for all neurons getting rid off one neuron parameter. Previous work by Date et. al [19] showed the Turing-completeness (TC) of SNNs. They used the equivalence between $\mu$-recursive functions and Turing Machines to prove TC of SNNs. Our work relaxes the requirements for TC to: tunable leak, weighted synapses and recursive connections.

### 4.3 Neuromorphic Application: MNIST Classification

MNIST [43] is a classical Machine Learning dataset consisting of 70,000 grayscale images. Each image is a $28 \times 28$ grid containing one handwritten digit. This dataset is commonly used to test the ability of new Machine Learning models for classification, and as a bench-test for new hardware architectures.

A simple and popular feed forward network used for image classification on MNIST is LeNet [42]. LeNet can be implemented in virtually any neural network framework, including Doryta and Whetstone [66]. Whetstone allows us to train restricted ANNs which can be easily encoded into memoryless SNNs. An ANN model trained with Whetstone can be encoded into memoryless SNNs with little modification. We follow the same crossbar structure as proposed by [53] with crossbar parameters as seen in Table 3(b). Notice that MNIST contains only 10 classes, while we use a total of 100 neurons (and thus an output of 100). This discrepancy is due to a loss

Table 3. Crossbar configurations for: (a) Conway's Game of Life (layer 1 receives spikes from layers 2, 3 and the initial input); (b) LeNet as crossbar connection. Boxes indicate crossbar units; lines represent connections between units and input/output.

(a)

| Layer | Input Lines | Number of Neurons | Synapses per Neuron |
|---|---|---|---|
| Board | 1200 | 400 | 3 |
| Life | 400 | 400 | 8.41 |
| Kill | 400 | 400 | 8.41 |

(b)

| # | Type | Input Lines | Filters | Number of Neurons | Synapses per Neuron |
|---|---|---|---|---|---|
| 1 | Conv | 784 | 1 | 784 | 1 |
| 2 | Conv | 784 | 6 | 784 | 22.90 |
| 3 | Conv | 784 | 6 | 196 | 4 |
| 4 | Conv | 1176 | 16 | 100 | 150 |
| 5 | Conv | 100 | 16 | 25 | 4 |
| 6 | Full | 400 | - | 120 | 400 |
| 7 | Full | 120 | - | 84 | 120 |
| 8 | Full | 84 | - | 100 | 84 |

incurred when enforcing the step function as the only activation function for the network (Whetstone's primary restriction).

The translation from a ANN trained in Whetstone, to a LIF-based SNN requires setting up some parameters. The spiking neuron parameters we have selected are: $V_e$ of 0, $V_{reset}$ of 0, $C$ of $1/256$, $R$ of 1 and the $V_{th}$ (threshold) is equal to the bias of the neuron (plus 0.5 as per details of Whetstone's implementation). The synapses' weights are in the same the weights as those obtained in Whetstone.

*4.3.1 Validating Doryta against Whetstone.* Using the neuron parameters above mentioned, we loaded in Doryta all models previously trained in Whetstone. We fed all models all $10,000$ test images. Whetstone's output and Doryta were compared spike per spike. No differences in the output of either tool was found. Even though both libraries/programs run in different languages, Doryta is capable of reproducing Whetstone's results.

Regarding simulation speed, however, Whetstone is a clear winner. Whetstone takes seconds to infer the class of all $10,000$ images while Doryta runs for a couple of minutes. This discrepancy is due to the high granularity of neuron operations in Doryta (a neuron is composed of several values), while Whetstone treats neurons as a summation boxes with no memory of past events. We expect that once transferred to hardware, memoryfull neurons will operate independently and in parallel of each other at much higher rates than what Doryta can simulate. As we show in Section 5.1, inferencing a single image requires only 8 clock cycles (one clock cycle per layer) which in spintronic hardware could take up to only a couple of microseconds. All models and scripts to check Doryta and Whetstone's outputs can be found in the supplementary material.

## 5 Experimental Results

In this section we present two sets of experiments: hardware performance estimation of a CNN classifier, and scalability of Doryta using GoL for benchmarking. Most experiments in this work were run using up to 32 compute nodes on RPI's AiMOS supercomputer [15]. Each node of AiMOS is composed of two, 20-core IBM

Power9 processors clocked at 3.15GHz and 512GiB of RAM. Doryta was compiled using IBM's `XLC_r` and the Spectrum MPI library. Additional experiments to determine the cache hit rate were run in a 20 core Intel Xeon Gold 5218R processor and made use of Intel's Performance Counter Monitor (PCM) tool to collect low-level hardware performance counter statistics. All experiments ran under spike-driven mode (Subsection 4.1.2).

## 5.1 Hardware Performance Estimation

Following Nikonov and Young's energy estimation strategy [53], we designed and trained four crossbar models for the task of image recognition. The models were tuned to process $28 \times 28$ black and white images from two datasets: MNIST and fashion-MNIST, and follow LeNet, as presented in Table 3(b). In their framework, Nikonov and Young assume an approximate layout for each component of the model in the chip, an approximate placement for each neuron, synapse and interconnect in each crossbar. In contrast, we gathered the usage of each component (leak, fire and integrate) and from these statistics, we calculated latency and energy (additionally to the area of the chip). This performance estimation process was automated in Python[3].

The four workloads (corresponding to four models) are:

**SL** (Small LeNet): a LeNet-like model trained on the MNIST dataset. It only differs from the original LeNet on the activation functions and the number of output neurons (100 for us, 10 traditionally).

**SLF** (Small LeNet Fashion): same model configuration as SL, but trained on Fashion-MNIST dataset [80].

**LL** (Large LeNet): model trained in MNIST dataset, but larger than SL. We increased the number of filters for the convolutional layers: 16 and 48 filters, instead of 6 and 32, respectively.

**LLF** (Large LeNet Fashion): same model configuration as LL but trained on Fashion-MNIST.

It is worth mentioning that, according to the output voltage of AFM neurons we adopted, sense amplifiers are required as part of the neuron to generate an appropriate signal. The absence of sense amplifier in this benchmark overestimates the performance of spintronics based networks.

The performance of the chip is divided into four distinct components: neurons, synapses, neuron-connected interconnects, and synapse-connected interconnects. The performance of each component, as well as the layer-wise performance, are presented in Figure 7 for both $Mn_3Sn$-based and NiO-based networks with 20 nm-width wire.

Figure 7(a) displays the energy consumption of each chip component. The $Mn_3Sn$-based network demonstrates a significant advantage over the NiO-based network, not only for neurons but also for synapses and their connected interconnects. This is due to the low input charge current, which enables lower operating voltages for both neurons and synapses. The large quantity of synapses makes the synapses and relevant interconnects the dominant term in the energy consumption. From Eq. 5, the operation voltage of synapses is proportional to both its resistance and neuron's input current, which accounts for $Mn_3Sn$-based network's superiority than NiO-based network in synapses.

Figure 7(b) shows the energy dissipated in each layer, which is dependent on workload allocation. The SLF workload has approximately 40% more integration and firing operations but the energy consumption is similar to the SL workload. Also, the LLF's integration and firing are 1.8 times that of LL's (see Figure 11) but less than 1.5 of the energy consumption. The fourth and sixth layer are dominant in energy consumption and the 'Fashion' workloads do not consume much more in these two layers so they are more energy efficient.

As mentioned in Equation 14, the latency of each layer is the collaborative contribution of one synaptic operation and one neuron operation, and it is dominated by the number of layers and interconnect delay per core. The area of each layer, dominated at the chip-level by the interconnect, is the largest contribution to the delay difference between different workloads implemented using the same neuron device. Figure 7(d) shows the latency of processing single layers. Note that only the network size, the number of cores and their size, influence

[3]The Python script can be found in the supplementary material.

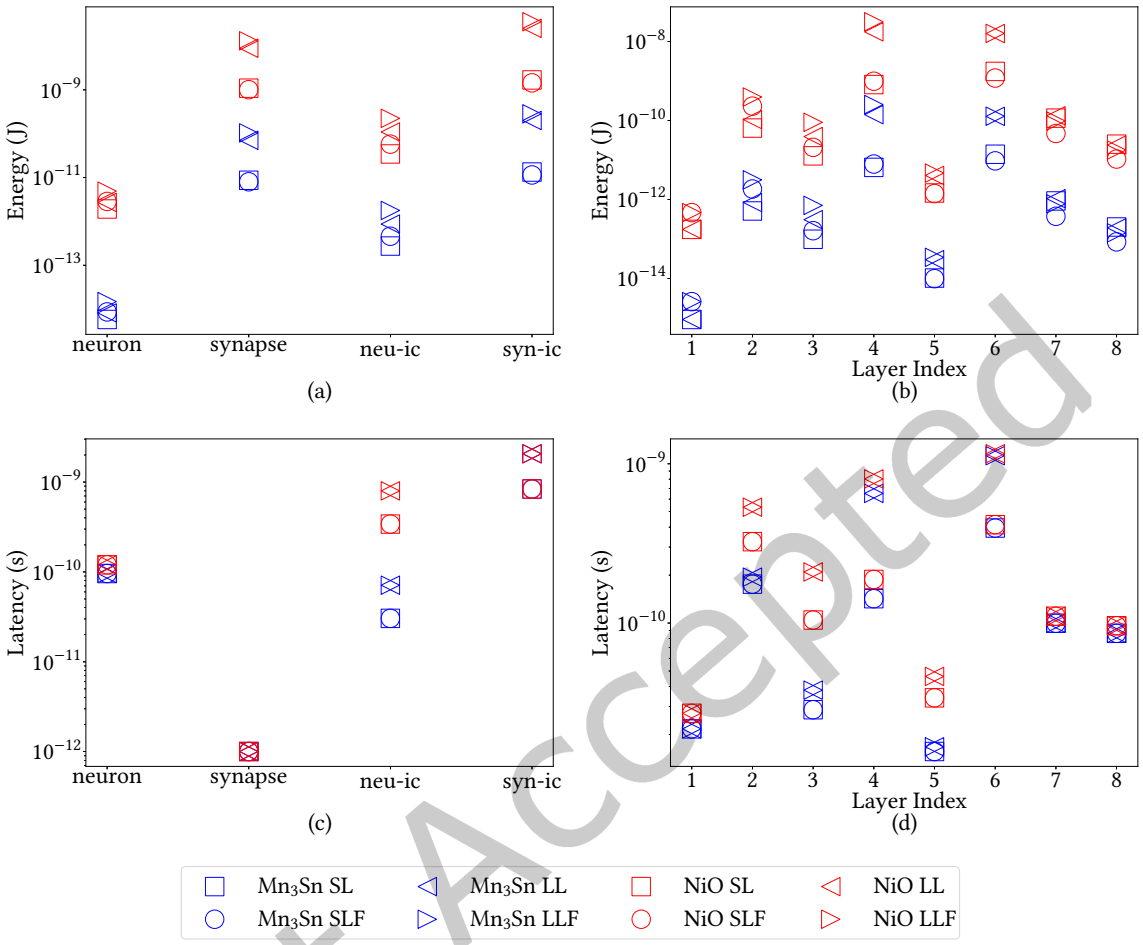



Fig. 7. (a) Energy consumed by different components. 'neu-ic' ('syn-ic') is the chip-level (core-level) interconnect. (b) Energy consumption reported layer by layer. (c) Latency of devices and interconnects. (d) Latency of each layer.



the latency of each layer. There is no difference latency-wise between workloads running on the same network model (e.g., no difference in latency between SL and SLF).

Figure 7(c) shows the latency of each component of the network. The dominant term is the synapse-connected interconnects and it is invariant regardless of neuron type and workload. It is only related to the area of each core and the interconnect RC properties. The synapses and interconnects both play a significant role in both the energy consumption and the latency computation, therefore, they have become the bottlenecks of further reduction in dissipation and latency.

Figure 8 shows the dependence of the latency and energy consumption on the interconnect width. Consistent with Eq. 10, the network latency increases as the wire width shrinks because of higher surface and grain boundary scatterings of the interconnect, which increases the interconnects' RC delay. The energy dependence shown in Figure 8 is that the energy consumption has a slight positive correlation with the interconnect size because the capacitance of interconnects slightly increases with wider wire.

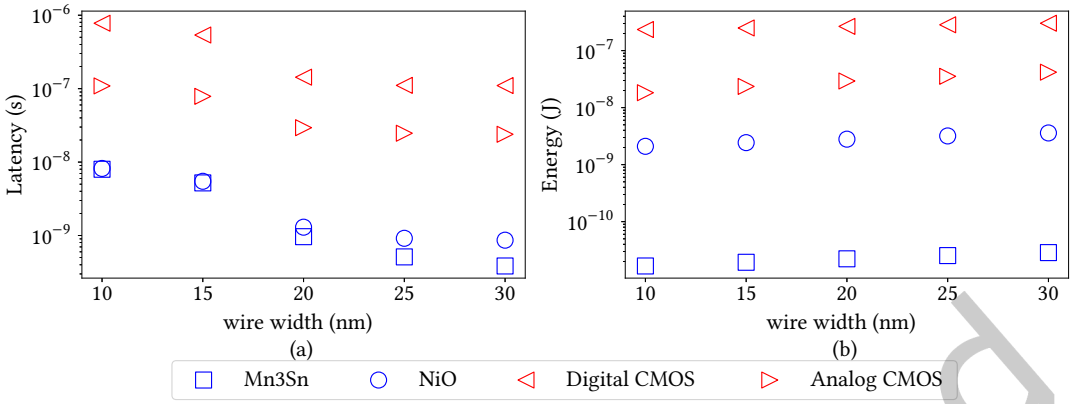



Fig. 8. (a) Latency of Small LeNet workload versus interconnect width. (b) Energy consumption of Small LeNet workload versus interconnect width for one inference.

Table 4 shows the performance estimate quantity per inference with the four different workloads for a 20 nm interconnect width. Benchmarks corresponding to analog and digital CMOS neural networks are also included as a comparison. Spintronics-based neural networks, especially $Mn_3Sn$-based network, show significantly superior performance compared with their CMOS counterparts. The AFM neural networks are promising to operate in higher frequency and take less chip area. For the best case,the AFM($Mn_3Sn$)-based network outperforms analog-CMOS network energy-wise by a factor of 1314× (= 29313 pJ/22.3 pJ) and latency-wise by a factor of 30× (= 29 ns/0.96 ns) and for the worst case, the AFM(NiO)-based network outperforms analog-CMOS network energy-wise by a factor of 10× (= 29313 pJ/2798 pJ) and latency-wise by a factor of 22× (= 29 ns/1.3 ns). When comparing with digital CMOS, the energy and latency improvements driven by $Mn_3Sn$-based devices are 11935× (= 266160 pJ/22.3 pJ) and 149× (= 143 ns/0.96 ns) respectively.

We also report the compound metric energy-delay product (EDP) for both spintronics-based and CMOS-based networks. We find that $Mn_3Sn$-based neural networks offer four orders of magnitude lower EDP compared to CMOS neural networks ($43100\times = 862 \times 10^{-18}\text{s} \cdot \text{J}/0.02 \times 10^{-18}\text{s} \cdot \text{J}$) and NiO-based neural networks offer two orders of magnitude lower EDP compared to CMOS neural networks ($237\times = 862 \times 10^{-18}\text{s} \cdot \text{J}/3.64 \times 10^{-18}\text{s} \cdot \text{J}$).

To compare these results to real world data, we took a look at the energy performance of a neuromorphic chip implementation, IBM's TrueNorth chip. According to Cheng et al. [17], TrueNorth could inference images from the MNIST dataset at a rate of 1249 frames per second with an energy performance of 6122.44 frames per second per Watt. Given that 1 Watt equals 1 J/s, 6122.44 frames/s/W is the same as 6122.44 frames/J or 163.334 $\mu$J/frame (163, 334 nJ/frame). One might be tempted to compare this energy estimate, 163, 334 nJ/frame, with any of the estimates on Table 4, but one must be careful because the size and shape of the models studied on both works are different. Cheng et al. based their work on the CIFAR network, ours is based on LeNet. Additionally, TrueNorth was conceptualized as a collection of $256 \times 256$ crossbars (called cores), while we assume variable size crossbars up to a size of $784 \times 784$. We approximate that the number of TrueNorth cores necessary to implement our largest network (LLF) is at least 1400. Given that Cheng et al. used 4064 cores for their network, an appropriate factor to compare their estimation to ours is 3 (divide 4064 by 1400). This means that inferencing an image using the TrueNorth chip with a model based on LeNet would take around 54, 444 nJ, while our estimation is of 4650 nJ for Digital CMOS. This one order of magnitude discrepancy can be explained by three factors: one, the lack of precise figures to compute the approximation; two, the myriad of CMOS implementations for synapses and neurons; and

Table 4. Performance of various technologies on different workloads. Note that the iso-latency power dissipation of various networks will be directly proportional to their energy consumption and is therefore not reported specifically in the table.

| Neuron | Workload | Area ($mm^2$) | Latency (ns) | Energy (pJ) | E· $\tau$ ($10^{-18}$s· J) |
|---|---|---|---|---|---|
| $Mn_3Sn$ | SL | 0.27 | 0.96 | 22.3 | 0.02 |
| | SLF | 0.27 | 0.96 | 19.9 | 0.02 |
| | LL | 1.55 | 2.23 | 272 | 0.61 |
| | LLF | 1.55 | 2.23 | 389 | 0.87 |
| NiO | SL | 0.27 | 1.3 | 2800 | 3.64 |
| | SLF | 0.27 | 1.3 | 2500 | 3.25 |
| | LL | 1.55 | 2.98 | 34100 | 102 |
| | LLF | 1.55 | 2.98 | 48800 | 145 |
| Analog CMOS | SL | 3.46 | 29.4 | 29300 | 862 |
| | SLF | 3.46 | 29.4 | 25900 | 762 |
| | LL | 19.7 | 49.2 | 414000 | 20400 |
| | LLF | 19.7 | 49.2 | 588000 | 28900 |
| Digital CMOS | SL | 38.2 | 143 | 266000 | 38100 |
| | SLF | 38.2 | 143 | 243000 | 34800 |
| | LL | 204 | 328 | 2670000 | 875000 |
| | LLF | 204 | 328 | 3870000 | 1270000 |

three, the cost associated with peripheral circuitry and transducers or amplifiers that will be needed, which we do not account for in our calculations.

Even though the performance results for spintronic neural networks are exciting, spintronic devices have some limitations compared to ideal LIF neurons. First, the synaptic weights can only be non-negative. We found that enforcing non-negative constraints on the weights resulted in trickier to train SNN models, yet, once trained, the workloads were not substantially different. Secondly, neuron leak and threshold cannot be tweaked as they are intrinsic values of the materials. Fortunately, when only considering positive weights, we only need to scale up or down the synaptic weights to keep the neuron threshold fixed. Lastly, spintronic synapses have about 64 different levels (6 bits), which—compared to the SNNs trained for this work—is a fraction of what single floating-point numbers can store. We found that restricting the network to unsigned 8-bit numbers did not affect accuracy by more than half a percent. It has been shown that 4 bits are enough for NNs to learn [72]. Thus, with the right technique to discretize the network, 6 bit networks should perform on par. We also found improvements on accuracy (up to 10%) with gray-scale images (instead of black and white) via temporally encoding spikes.

## 5.2 Parallel Performance Experiments - Strong Scaling

As seen in Table 5, a strong scaling experiment was performed using GoL with a grid of size $1024 \times 1024$, starting on one core (or MPI rank) and scaled up to 1024 cores on 32 nodes. Note, that only 32 of the 40 cores available on each IBM Power9 compute node were used. The other 8 cores were made available for OS and other system jobs and to reduce the effects of any OS jitter [3]. The parallel simulation ran for 1000 GoL steps starting with the same initial configuration (each cell had a probability of 20% of being alive). A total of 3, 145, 728 LPs are required for the simulation (3 neurons per cell) and $1.88 \times 10^9$ events were processed. Because of Doryta's determinism, all executions generated the same number of commited events ($1.88 \times 10^9$). After carefully tuning all ROSS simulation parameters (number of KPs per PE, batch size, and GVT parameters), it was observed that an optimistic event

Table 5. Strong scaling for a simulation of GoL using the tie-breaker mechanism. Randomly initialized GoL grid with size of $1024 \times 1024$. Sequential execution time of 5230.72 s. Conservative mode execution with batch size of 512 and GVT interval of 128. Optimistic simulation restricted to a maximum lookahead of $\frac{1}{2}\Delta t$, with a batch size of 32, and 128 KPs per PE.

| | | Conservative | | Optimistic | |
|---|---|---|---|---|---|
| Nodes | Cores | Runtime (sec) | Speedup | Runtime (sec) | Speedup |
| 1 | 2 | 3084.18 | **1.70** | 3224.01 | 1.62 |
| 1 | 4 | 1397.29 | **3.74** | 1457.20 | 3.59 |
| 1 | 8 | 648.34 | **8.07** | 679.55 | 7.70 |
| 1 | 16 | 315.65 | **16.57** | 332.94 | 15.71 |
| 1 | 32 | 157.65 | **33.18** | 165.90 | 31.53 |
| 2 | 64 | 77.33 | **67.64** | 81.23 | 64.39 |
| 4 | 128 | 40.81 | 128.16 | 40.53 | **129.06** |
| 8 | 256 | 23.55 | 222.09 | 21.43 | **244.08** |
| 16 | 512 | 18.12 | 288.75 | 13.98 | **374.11** |
| 32 | 1024 | 16.37 | **319.58** | 16.87 | 310.00 |

scheduler is nearly as fast as the conservative event scheduler, and in some cases even faster. Nonetheless, as seen in Tables 6, 7 and 8, the conservative approach still runs faster than optimistic in most circumstances.

Doryta performs much better with the conservative approach because of the intrinsic lookahead present in the model. Every single neuron fires at the same virtual time (all heartbeat events are scheduled at the same virtual timestamps by each LP independently) and each neuron schedules all spikes events for the same virtual time slot in the future. Because this virtual time slot is encoded as $\frac{1}{2}\Delta t$ from any heartbeat, the simulation advances at $\frac{1}{2}\Delta t$-steps virtual time increments. With the optimistic approach, a PE might be too "optimistic" and might run ahead too far of this time increment, thus it must rollback all events that were processed outside the time increment. The solution is to constrain optimism with a maximum lookahead equal to half the virtual time increment, i.e, an optimistic lookahead of $\frac{1}{2}\Delta t$ (see Elastic Time [70]). This constraint forces the optimistic simulation to increment at the same virtual times as the conservative simulation does, which makes optimistic nearly as fast as conservative as seen in Tables 5-8. Every virtual increment step triggers a GVT operation in both modes. Optimally, the minimum number of GVT operations to be performed in 1000 GoL steps is 4000 (4 per GoL step). Most conservative simulations utilized 4002 GVT operations while most optimistic 4001. This means that both conservative and optimistic approaches consume the same number of events per GVT operation.

Other network configurations such as a fully connected layer might not scale as nicely. The GoL configuration scales well because the number of connections each neuron has is small (to its neighbours and itself, 9 at most). This sparse fan-out feature is an advantage of 2D convolutional connections as opposed to all-to-all connections.

To quantify the performance impact of the tie-breaker mechanism —which guarantees a deterministic simulation— we collected parallel simulation performance data for models with and without the tie-breaker enabled as shown in Table 6. If the initial spike events (determined by the user) do not overlap with any heartbeat event, then the result of the simulation will be deterministic, i.e., the order in which tied events are processed does not matter in Doryta, in as much they are the same kind of event. We observe that the simulation slows by a factor of 7 (in serial execution) when the tie-breaker mechanism is turned on. However, the simulation is only slowed down by a factor of 1.6 in parallel. This difference between speedups of serial vs parallel hints to the idea that it is the network which keeps the simulation from running faster.

In general, for any ROSS parallel simulation, if a model does not require the added deterministic guarantee of the tie-breaker mechanism, it is recommended to disable it and save memory and execution time.

Table 6. Strong scaling for a simulation of GoL with tie-breaker mechanism **deactivated**. Randomly initialized GoL grid with size of $1024 \times 1024$. Sequential execution time of 674.49 s. Conservative mode execution with batch size of 512 and GVT interval of 512. Optimistic mode restricted to a maximum lookahead of $\frac{1}{2}\Delta t$, with a batch size of 64 and 16 KPs per PE.

| | | Conservative | | Optimistic | |
|---|---|---|---|---|---|
| Nodes | Cores | Runtime (sec) | Speedup | Runtime (sec) | Speedup |
| 1 | 2 | 687.09 | **0.98** | 850.91 | 0.79 |
| 1 | 4 | 406.35 | **1.66** | 485.64 | 1.39 |
| 1 | 8 | 210.84 | **3.20** | 250.43 | 2.69 |
| 1 | 16 | 109.67 | **6.15** | 130.02 | 5.19 |
| 1 | 32 | 56.92 | **11.85** | 66.96 | 10.07 |
| 2 | 64 | 27.90 | **24.17** | 33.48 | 20.15 |
| 4 | 128 | 15.48 | **43.58** | 17.36 | 38.86 |
| 8 | 256 | 10.25 | **65.82** | 11.06 | 61.01 |
| 16 | 512 | 11.28 | **59.77** | 11.47 | 58.82 |
| 32 | 1024 | 12.10 | **55.73** | 14.18 | 47.57 |

Table 7. Strong scaling experiments with tie-breaker enabled using GoL with a variable grid size of 1024 (1K), 2048 (2K), 4096 (4K) and 8192 (8K). All simulations used the same parameters as in Table 5

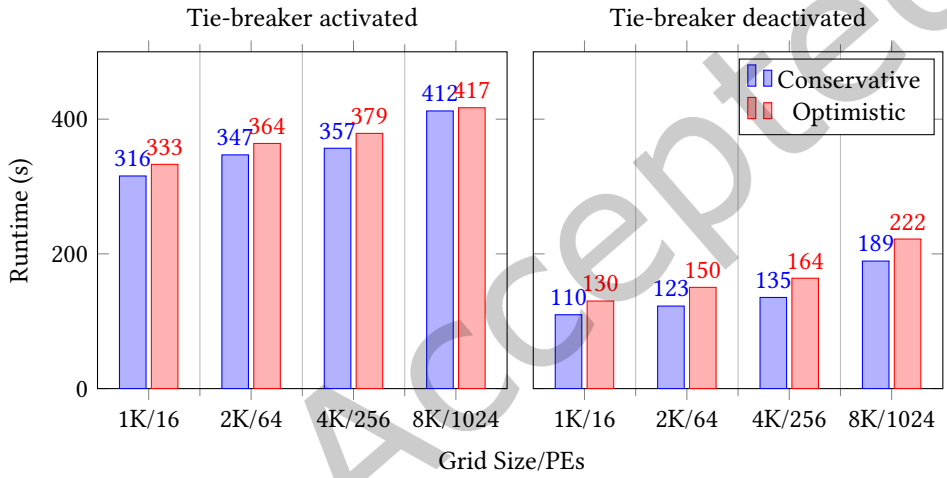
| | Conservative - Runtime (s) | | | | | Optimistic - Runtime (s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Grid size | | | | | Grid size | | | |
| Nodes | Cores | 1K | 2K | 4K | 8K | Cores | 1K | 2K | 4K | 8K |
| 1 | 32 | **156.53** | **727.44** | **3369.99** | | 32 | 165.40 | 760.66 | 3544.26 | |
| 2 | 64 | **77.33** | **346.97** | **1560.51** | **7348.29** | 64 | 81.23 | 363.97 | 1652.48 | |
| 4 | 128 | 40.81 | **166.46** | **737.73** | **3348.39** | 128 | **40.53** | 175.18 | 783.15 | 3612.13 |
| 8 | 256 | 23.55 | 87.35 | **356.83** | **1583.76** | 256 | **21.43** | **87.21** | 378.90 | 1701.34 |
| 16 | 512 | 18.12 | 53.36 | **187.59** | **752.83** | 512 | **13.98** | **47.44** | 188.71 | 817.52 |
| 32 | 1024 | **13.50** | 38.73 | 113.11 | **412.18** | 1024 | 14.71 | **34.37** | **110.95** | 416.95 |

To reduce the overall noise in performance data, we ran the experiments again in the larger grid size configurations as shown in Table 7. A clear trend can be seen for the 1K case. Optimistic performs better when scaled to 128, 256 and 512 cores. This trend does not appear in higher grid sizes (2K, ...). A variation of this trend appears as the number of cores increases so that optimistic closes the runtime gap on conservative. At a certain point optimistic reaches its maximum speedup while conservative takes more cores to reach its maximum speedup.

For the 1K case on conservative, the jump from 32 to 64 cores more than doubles in relative speedup (2.02), but this does not happen from 64 to 128 (1.89). Thus, the 1K case is the maximally speed-up on 64 cores. For the optimistic case, the jump from 64 to 128 still more than doubles the speedup (2.004). It is precisely because of the sustained speedup gain by optimistic that it is capable of closing the gap and runs faster at 128 cores and up. The same behaviour can be seen in the larger cases: the jump from 128 to 256 cores in the 2K grid size more than doubles the speedup for optimistic (2.01) but does not for conservative (1.91); the jump from 256 to 512 cores in the 4K case results in a speedup of 2.01 for optimistic and 1.90 for conservative; and although it does not more than double the speedup, the jump from 512 to 1024 cores in the 8K case shows a larger positive speedup for optimistic (1.96) than for conservative (1.83).

Table 8. Scaling-up experiments to a GoL with grid size of 1024 (1K), 2048 (2K), 4096 (4K) and 8192 (8K). Tie-breaker off. All runs used the same parameters as in Table 6

| | Conservative - Runtime (s) | | | | | Optimistic - Runtime (s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Grid size | | | | | Grid size | | | |
| Nodes | Cores | 1K | 2K | 4K | 8K | Cores | 1K | 2K | 4K | 8K |
| 1 | 32 | 53.99 | 244.05 | 1046.32 | 4444.65 | 32 | 68.32 | 299.67 | 1265.32 | 5280.50 |
| 2 | 64 | 27.90 | 122.55 | 518.54 | 2188.58 | 64 | 33.48 | 150.32 | 624.01 | 2618.48 |
| 4 | 128 | 15.48 | 62.82 | 261.94 | 1090.15 | 128 | 17.36 | 75.46 | 314.54 | 1311.06 |
| 8 | 256 | 10.25 | 34.99 | 135.33 | 548.34 | 256 | 11.06 | 41.13 | 163.88 | 658.24 |
| 16 | 512 | 11.28 | 25.91 | 79.24 | 291.30 | 512 | 11.47 | 28.97 | 93.75 | 349.90 |
| 32 | 1024 | 12.10 | 27.06 | 61.60 | 189.33 | 1024 | 14.18 | 29.92 | 72.88 | 222.02 |

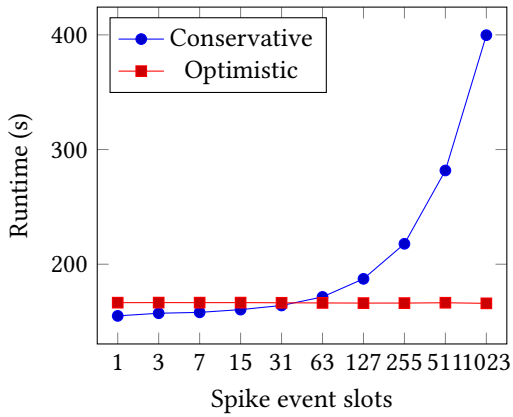



| Grid size | 1K/16 | 2K/64 | 4K/256 | 8K/1024 |
|---|---|---|---|---|
| % remote events | 0.640 | 1.278 | 2.597 | 5.203 |

Fig. 9. Weak scaling on 16, 64, 256 and 1024 MPI ranks. Left, tie-breaker enabled; right, tie-breaker disabled.

Disabling the tie-breaker mechanism erases optimistic's performance advantage, see Table 8. Notice that a 2K grid is four times larger than a 1K grid, thus the simulation should take four times as much, but at 1024 cores the 2K grid takes less than 3 times the 1K grid time, while at 32 cores is larger than 4 times. With 32 cores, the 8K grid takes 82.33 times what the 1K grid takes to run, although it should be 64. This suggests that 1K case is far too small to fully utilize 1024 cores.

Cutting across the strong scaling-up experiments from Tables 7 and 8, we can derive two sets of weak scaling experiments, see Figure 9. The largest experiment, 8K, utilized a total of 200 million neurons (# cells $\times$ # neurons per cell = $8192^2 \times 3$) and simulated a total of 2000 virtual clock steps (heartbeat steps) in 412.18 seconds with the tie-breaker mechanism activated, and 222.02 seconds without tie-breaker. Conway's Game of Life weakly scales with very little overhead (around 5% to 15%) except for the 8K/1024 non tie-breaker case, which might have been an exception (due to possible interference with other jobs scheduled at the supercomputer). The overhead of 5 to 15% is partially due to the doubling of neighboring connections relative to total neurons per between MPI




| Slots | Conservative | | Optimistic | |
|---|---|---|---|---|
| | Runtime | GVT ops | Runtime (s) | GVT ops |
| 1 | 154.96 | 4077 | 166.48 | 4001 |
| 3 | 157.26 | 8000 | 166.54 | 4001 |
| 7 | 158.02 | 15996 | 166.51 | 4001 |
| 15 | 160.41 | 31988 | 166.52 | 4001 |
| 31 | 164.01 | 63972 | 166.37 | 4001 |
| 63 | 171.49 | 127940 | 166.24 | 4001 |
| 127 | 187.26 | 255876 | 166.14 | 4001 |
| 255 | 217.82 | 511748 | 166.15 | 4001 |
| 511 | 281.76 | 1023492 | 166.42 | 4001 |
| 1023 | 399.72 | 2046980 | 165.88 | 4000 |

Fig. 10. Thinning intrinsic lookahead present in Doryta models. The number of slots indicates the number of possible virtual times between two heartbeats that a spike event can be scheduled to.

Table 9. Performance data across four possible configurations on 64 cores: conservative vs optimistic modes, and tie-breaker activated vs deactivated.

| Stat | Tie-breaker | | No tie-breaker | |
|---|---|---|---|---|
| | Conservative | Optimistic | Conservative | Optimistic |
| Events processed | 1875038719 | 1875039282 | 1875038719 | 1875038719 |
| PE event ties | 0 | 0 | 1875038719 | 1875038719 |
| Events rolled back | - | 563 | - | 0 |
| Runtime (s) | 77.33 | 81.23 | 27.90 | 33.48 |
| Num GVT calculations | 4002 | 4001 | 4002 | 4001 |
| GVT time (s) | 28.34 | 29.30 | 13.54 | 16.92 |
| Fossil collect time (s) | - | 6.34 | - | 4.00 |
| Event processing time (s) | 44.38 | 45.18 | 11.07 | 11.47 |



ranks. This means that number of remote events per MPI rank doubles as the width of the rows increases by a factor of two (The 1K/16 and 2K/64 cases have the same number of neurons/LPs, but the second has twice the number of connections to other PEs).

Conservative outperforms optimistic because of the intrinsic lock-stepped fashion in which the NN model advances with at most two GVT operations to finish a neuron cycle. In other words, conservative is capable of advancing the simulation between two GVT operations as efficiently as optimistic can when no backtracking is needed. Figure 10 shows how reducing the amount of work performed in between GVT operations erases conservative's advantage over optimistic. For this, instead of scheduling spike events precisely in between heartbeat events, i.e, shifted by $\frac{1}{2}\Delta t$ of heartbeats, we schedule them to $\frac{1}{2n}\Delta t$ increments of the heartbeats, where $n \in \mathbb{Z}^+$. When $n = 2$, there are 3 slots in which a spike could be scheduled (now $+\frac{1}{4}$, now $+\frac{2}{4}$ and now $+\frac{3}{4}$). The more slots there are, the more GVT operations conservative has to perform. At 1023 slots ($n = 10$), conservative performs $2 \times 10^6$ GVT operations.

Table 10. Cache memory performance data for conservative and optimistic modes running in Intel Xeon Gold 5218R. Cache hits are always higher for conservative than optimistic.

| | Conservative | | | | Optimistic | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Cores | Runtime (s) | Event Processing (s) | L2 Hit Rate | L3 Hit Rate | Runtime (s) | Event Processing (s) | Fossil Collect (s) | L2 Hit Rate | L3 Hit Rate |
| 2 | 680.07 | 555.07 | 44% | 44% | 767.23 | 574.93 | 82.13 | 39% | 38% |
| 4 | 334.99 | 268.66 | 46% | 38% | 381.23 | 276.55 | 43.05 | 41% | 32% |
| 8 | 167.60 | 131.11 | 49% | 31% | 190.91 | 135.37 | 22.54 | 43% | 24% |
| 16 | 88.99 | 67.52 | 52% | 25% | 102.12 | 70.26 | 12.17 | 46% | 20% |

Table 11. Average total number of integration and fire operations utilized on the inference of a black & white, $28 \times 28$ image from MNIST and Fashion-MNIST datasets. Accuracy of each network on their respective dataset: Small and Large LeNet on MNIST and Fashion-MNIST.

| Workload | Integration | Fire | Accuracy |
|---|---|---|---|
| Small LeNet | 73734.31 | 733.94 | 96.36% |
| Small LeNet Fashion | 104355.97 | 1180.18 | 67.12% |
| Large LeNet | 296092.07 | 965.04 | 98.14% |
| Large LeNet Fashion | 534441.86 | 1793.55 | 70.06% |

One might think that conservative outperforms optimistic because of rollbacks, but this hypothesis is disproved once we look at runtime with the tie-breaking mechanism turned off, see Table 9. Notice that optimistic never rolls back and performs one less GVT operation than conservative, yet it is significantly slower. Observe that fossil collect and event processing times are the main culprits for optimistic's slower runtime. Table 10 shows that the cache hit rate on an Intel Xeon 20 core processor for both optimistic and conservative event scheduling approaches. We observe that conservative has a higher L2 and L3 cache hit rate than optimistic, which results in slower execution time for optimistic event processing. The conservative approach's cache performance behavior is attributed to memory used by events being reclaimed immediately after events are processed, while in optimistic event processing, memory is only reclaimed after GVT is performed during fossil collection. We hypothesize most of the runtime difference to be a result of cache misses.

## 6 Related Work

Our performance estimation of spintronic materials work builds upon Nikonov and Young's [53], but instead of assuming a generic SNN workload, we have generated workloads via simulation. Through this, we have been able to observe that different workloads (MNIST vs Fashion-MNIST) do not have significant device performance differences, although different workloads are associated with models with different accuracy. We confirmed the suspicion that the shape of the network has the largest impact on performance. Larger models (e.g., more layers) use more energy, space and have higher latencies. Additionally, we also studied the effects of interconnect wire width and found an optimal size for our scenarios.

The spintronic properties of device materials are good emulators of spiking neurons, yet others device types exist. These include memristors [63, 82] and photonics [34, 73]. What makes antiferromagnetic materials attractive is the very low critical currents at which they can be excited and the possibility to build them with existing technologies. An important observation from our performance estimation analysis is that interconnects take up the bulk of the energy consumption of the network. A similar observation has been found in biological brains. In

mammalian brains, communication between neurons consumes one to two orders of magnitude more energy than computation itself [44].

Since the advent of neuromorphic computing in 1990 [48], research in the area has grown to thousands of articles [64] ranging from theory [10] and simulation (Brian [30] and NEST [32]) to production hardware implementations by established chip manufactures like Intel and IBM with the Loihi [20] and TrueNorth [14] chips, respectively. A key research challenge arises from these novel chips, namely: how to address future design trade-offs as well integrate them into larger computing systems? A central approach for addressing this challenge is modeling and simulation [9, 79].

NeMo[4] [61] is an early massively parallel neuromorphic simulation framework. As mentioned in Section 4, NeMo's ability to simulate the TrueNorth chip makes it inflexible. While NeMo and Doryta are both built as PDES models, utilizing the ROSS parallel simulation engine [11], only Doryta takes advantage of the lock-stepped synchronization of SNNs to speedup the simulation by constraining optimistic lookahead which can be seen as a special case of the Elastic Time algorithm [71]. Discrete-event simulation (DES) of spiking neurons is not a recent development. Watts [78] observed that the performance of spiking neuron simulations could be improved if event-driven simulation was used instead of a purely time-stepped approachd. We observe a similar speed-up gain in Doryta simulations when using Doryta's spike-driven mode. DES has not only been used to speed-up simulations but also to increase the accuracy of simulation results. Such is the case with Pimpini et al. [59], who developed a PDES simulation in ROOT-Sim [56] (ROme OpTimistic Simulator, a Time Warp-based simulator framework) with the express purpose of achieving a higher simulation accuracy. However, a key departure from their work and ours is that Doryta was built as a tool to trace neuron behaviour such as firing of synapses. Additionally, Boulet et al. [5] implemented N2S3 which is an SNN simulator written in Scala using an actor model-based library. DES and the actor model are similar in that the model computation is divided into pieces and is advanced through the use of messages, yet the actor model lacks the notion of virtual time which is central to advancing DES models computations. We additionally observe that N2S3 runs on top of the Java Runtime Environment without any parallelization library such as the use of MPI for communication, making it unsuitable for execution on modern HPC systems.

Although plenty of SNN models could be simulated by Doryta, loading them requires custom made handlers. We have implemented a translation procedure to take models trained with Whetstone [66] into a binary format that Doryta understands. Building custom translation procedures is expensive. An approach to make Doryta more accessible is to expose it as a backend SNN simulator for other frameworks such as PyNN [21]. This is precisely what Pimpini [58] is working on their ROOT-Sim-based SNN simulator. We consider PyNN integration future work.

## 7 Conclusion

In an analysis of two spintronic-neuron models, $Mn_3Sn$ and NiO, we found that the bulk of the energy consumption of the chips would be driven by the connection between neurons and not the neurons themselves, the interconnect. Compared to CMOS, our analysis indicates that spintronic-based chips have an energy-delay product that is **three to six orders of magnitude smaller** at inferencing a single image using LeNet. We have also presented Doryta, a parallel discrete-event-based, chip-agnostic simulator for neuromorphic applications, which we applied to the energy estimation of novel spintronic-based devices. We observed that Doryta can be scaled up to over 1,000 CPU cores. Doryta simulated a sparse 200 million neuron model for a total of 2000 virtual clock steps in under four minutes using 1,024 CPU cores. Additionally, Doryta is able to reproduce the inference results of another spiking-neural network library, Whetstone, with one key advantage: it takes into account time information

[4]Not to be confused with NeMo 2009 [23], a GPU-based simulation framework for SNNs.

and delays. It was through loading Whetstone's models into Doryta that could determine the workloads that spintronic-based chips would require in terms of basic operations.

As future work, we plan to incorporate the cost associated with peripheral circuitry and transducers or amplifiers that are needed in a working chip. We plan to look into ferromagnetic based spiking neurons due to their ease of fabrication and characterization in the GHz regime as opposed to THz for spintronics. Since we found that interconnects are the bottlenecks, we see an interesting avenue for research in the improvement of interconnects. Finally, as for Doryta, we intend to implement further non-ML applications using SNNs such as: digital circuits, RAM, and a fully-fledged computer digital computer; and, make Doryta a supported simulator by PyNN.

## Acknowledgments

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-21-1-1010. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. Additionally, computational time on the AiMOS supercomputer was provided by Rensselaer's Center for Computational Innovations. We thank the time and careful reading by the reviewers of the multiple versions of the manuscript. Without them, this paper would not be what it is.



## References

[1] H. Bauer and C. Sporrer. 1993. Reducing Rollback Overhead In Time-warp Based Distributed Simulation With Optimized Incremental State Saving. In *[1993] Proceedings 26th Annual Simulation Symposium*. IEEE, Arlington, VA, 12–20. https://doi.org/10.1109/SIMSYM.1993.639048

[2] D. W. Bauer Jr., C. D. Carothers, and A. Holder. 2009. Scalable Time Warp on Blue Gene Supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, Washington, DC, USA, 35–44.

[3] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, Susan Coghlan, and Aroon Nataraj. 2008. Benchmarking the effects of operating system interference on extreme-scale parallel machines. *Cluster Computing* 11, 1 (2008), 3–16. https://doi.org/10.1007/s10586-007-0047-2

[4] Elwyn R. Berlekamp, John Horton Conway, and Richard K. Guy. 1982. *Winning Ways for Your Mathematical Plays. 2: Games in Particular*. Academic Press, London.

[5] Pierre Boulet, Philippe Devienne, Pierre Falez, Guillermo Polito, Mahyar Shahsavari, and Pierre Tirilly. 2017. *N2S3, an Open-Source Scalable Spiking Neuromorphic Hardware Simulator*. Research Report. Université de Lille 1, Sciences et Technologies ; CRIStAL UMR 9189.

[6] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. 2007. Simulation of Networks of Spiking Neurons: A Review of Tools and Strategies. *J Comput Neurosci* 23, 3 (Dec. 2007), 349–398.

[7] R. E. Bryant. 1977. *Simulation of Packet Communication Architecture Computer Systems*. Ph. D. Dissertation. MIT.

[8] Yi Cao, AndrewW Rushforth, Yu Sheng, Houzhi Zheng, and Kaiyou Wang. 2019. Tuning a Binary Ferromagnet into a Multistate Synapse with Spin–Orbit-Torque-Induced Plasticity. *Adv Funct Mater* 29, 25 (2019), 1808104.

[9] Suma George Cardwell, Craig Vineyard, Willam Severa, Frances S. Chance, Frederick Rothganger, Felix Wang, Srideep Musuvathy, Corinne Teeter, and James B. Aimone. 2020. Truly Heterogeneous HPC: Co-design to Achieve What Science Needs from HPC. In *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI*, Jeffrey Nichols, Becky Verastegui, Arthur 'Barney' Maccabe, Oscar Hernandez, Suzanne Parete-Koon, and Theresa Ahearn (Eds.). Vol. 1315. Springer International Publishing, Cham, 349–365. https://doi.org/10.1007/978-3-030-63393-6_23

[10] Nicholas T. Carnevale and Michael L. Hines. 2009. *The NEURON Book* (1st ed.). Cambridge University Press, USA.

[11] Christopher D. Carothers, David Bauer, and Shawn Pearce. 2002. ROSS: A High-Performance, Low-Memory, Modular Time Warp System. *J. Parallel and Distrib. Comput.* 62, 11 (Nov. 2002), 1648–1669. https://doi.org/10.1016/S0743-7315(02)00004-7

[12] Christopher D. Carothers and Kalyan S. Perumalla. 2010. On deciding between conservative and optimistic approaches on massively parallel platforms. In *Winter Simulation Conference'10*. 678–687.

[13] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Trans. Model. Comput. Simul.* 9, 3 (1999), 224–253.

[14] Andrew S. Cassidy, Paul Merolla, John V. Arthur, Steve K. Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta, Jun Sawada, Theodore M. Wong, Vitaly Feldman, Arnon Amir, Daniel Ben-Dayan Rubin, Filipp Akopyan, Emmett McQuinn, William P. Risk, and Dharmendra S. Modha. 2013. Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 1–10.

[15] Center for Computational Innovations. [n. d.]. Artificial intelligence multiprocessing optimized system (AiMOS). cci.rpi.edu. https://cci.rpi.edu/aimos (accessed Feb 1, 2022).

[16] K. M. Chandy and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. In *IEEE Transactions on Software Engineering*, Vol. 24. 440–452.

[17] Hsin-Pai Cheng, Wei Wen, Chunpeng Wu, Sicheng Li, Hai Helen Li, and Yiran Chen. 2017. Understanding the Design of IBM Neurosynaptic System and Its Tradeoffs: A User Perspective. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, Lausanne, 139–144.

[18] Ran Cheng, Di Xiao, and Arne Brataas. 2016. Terahertz antiferromagnetic spin Hall nano-oscillator. *Physical review letters* 116, 20 (2016), 207603.

[19] Prasanna Date, Catherine Schuman, Bill Kay, and Thomas Potok. 2021. Neuromorphic Computing Is Turing-Complete. *arXiv:2104.13983 [cs]* (April 2021).

[20] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (January 2018), 82–99.

[21] Andrew P Davison. 2008. PyNN: A Common Interface for Neuronal Network Simulators. *Front. Neuroinform.* 2 (2008). https://doi.org/10.3389/neuro.11.011.2008

[22] Jianting Dong, Xinlu Li, Gautam Gurung, Meng Zhu, Peina Zhang, Fanxing Zheng, Evgeny Y. Tsymbal, and Jia Zhang. 2022. Tunneling Magnetoresistance in Noncollinear Antiferromagnetic Tunnel Junctions. *Phys. Rev. Lett.* 128 (May 2022), 197201. Issue 19. https://doi.org/10.1103/PhysRevLett.128.197201

[23] Andreas K. Fidjeland, Etienne B. Roesch, Murray P. Shanahan, and Wayne Luk. 2009. NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, Boston, MA, USA, 137–144. https://doi.org/10.1109/ASAP.2009.24

[24] I Fina, X Marti, D Yi, J Liu, JH Chu, C Rayan-Serrao, S Suresha, AB Shick, J Železnỳ, T Jungwirth, et al. 2014. Anisotropic magnetoresistance in an antiferromagnetic semiconductor. *Nature communications* 5, 1 (2014), 1–7.

[25] Daniel Gall, Judy J. Cha, Zhihong Chen, Hyeuk Jin Han, Christopher Hinkle, Joshua A. Robinson, Ravishankar Sundararaman, and Riccardo Torsi. 2021. Materials for interconnects. *MRS Bulletin* 46, 10 (Oct. 2021), 959–966.

[26] Jeff Gambino. 2018. Chapter 6 - Process Technology for Copper Interconnects. In *Handbook of Thin Film Deposition (Fourth Edition)* (fourth edition ed.), Krishna Seshan and Dominic Schepis (Eds.). William Andrew Publishing, 147–194. https://doi.org/10.1016/B978-0-12-812311-9.00006-2

[27] Martin Gardner. 1970. The Fantastic Combinations of Jhon Conway's New Solitaire Game "Life". *Sci. Am.* 223 (1970), 20–123.

[28] Wulfram Gerstner and Werner M. Kistler. 2002. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Cambridge, U.K.

[29] Ashok K. Goel. 2008. *Parasitic Resistances, Capacitances, and Inductances*. 46–135. https://doi.org/10.1002/9780470165973.ch2

[30] Dan Goodman and Romain Brette. 2008. Brian: A Simulator for Spiking Neural Networks in Python. *Frontiers in Neuroinformatics* 2 (2008).

[31] Julie Grollier, Damien Querlioz, KY Camsari, Karin Everschor-Sitte, Shunsuke Fukami, and Mark D Stiles. 2020. Neuromorphic spintronics. *Nature electronics* 3, 7 (2020), 360–370.

[32] Plesser Hans. 2008. NEST 2: A Parallel Simulator for Large Neuronal Networks. *Front. Neuroinform.* 2 (2008). https://doi.org/10.3389/conf.neuro.11.2008.01.137

[33] Jennifer Hasler and Harry Marr. 2013. Finding a roadmap to achieve large neuromorphic hardware systems. *Front Neurosci* 7 (2013).

[34] Matěj Hejda, Ekaterina Malysheva, Dafydd Owen-Newns, Qusay Raghib Ali Al-Taai, Weikang Zhang, Ignacio Ortega-Piwonka, Julien Javaloyes, Edward Wasige, Victor Dolores-Calzadilla, José M. L. Figueiredo, Bruno Romeira, and Antonio Hurtado. 2022. Artificial Optoelectronic Spiking Neuron Based on a Resonant Tunnelling Diode Coupled to a Vertical Cavity Surface Emitting Laser. https://doi.org/10.48550/arXiv.2206.11044 arXiv:2206.11044 [physics]

[35] Atsufumi Hirohata, Keisuke Yamada, Yoshinobu Nakatani, Ioan-Lucian Prejbeanu, Bernard Diény, Philipp Pirro, and Burkard Hillebrands. 2020. Review on spintronics: Principles and device applications. *J Magn Magn Mater* 509 (2020), 166711.

[36] Dazhi Hou, Zhiyong Qiu, Joseph Barker, Koji Sato, Kei Yamamoto, Saül Vélez, Juan M Gomez-Perez, Luis E Hueso, Felix Casanova, and Eiji Saitoh. 2017. Tunable sign change of spin Hall magnetoresistance in Pt/NiO/YIG structures. *Physical review letters* 118, 14 (2017), 147202.
[37] Tomoki Ikeda, Masakiyo Tsunoda, Mikihiko Oogane, Seungjun Oh, Tadashi Morita, and Yasuo Ando. 2018. Anomalous Hall effect in polycrystalline Mn3Sn thin films. *Applied Physics Letters* 113, 22 (2018), 222405.
[38] David Jefferson, Brian Beckman, Fred Wieland, Leo Blume, Mike Di Loreto, Phil Hontalas, Pierre Laroche, Kathy Sturdevant, Jack Tupman, Van Warren, John Wedel, Herb Younger, and Steve Bellenot. 1987. *Distributed simulation and the time warp operating system.* Technical Report OSTI 5639121. NASA Jet Propulsion Laboratory, Pasadena, CA, USA.
[39] David Jefferson and Henry Sowizral. 1985. *Fast concurrent simulation using the time warp mechanism.* Technical Report ADA129431. Rand. Corp., Santa Monica, CA, USA. Issue 2.
[40] David R. Jefferson. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7, 3 (1985), 404–425.
[41] Roman Khymyn, Ivan Lisenkov, James Voorheis, Olga Sulymenko, Oleksandr Prokopenko, Vasil Tiberkevich, Johan Akerman, and Andrei Slavin. 2018. Ultra-fast artificial neuron: generation of picosecond-duration spikes in a current-driven antiferromagnetic auto-oscillator. *Scientific reports* 8, 1 (2018), 1–9.
[42] Yann LeCun, Bernhard Boser, John S. Denker, Donnie. Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* 1, 4 (Dec. 1989), 541–551.
[43] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998. The MNIST Handwritten Digit Database. http://yann.lecun.com/exdb/mnist/.
[44] William B Levy and Victoria G. Calvert. 2021. Communication Consumes 35 Times More Energy than Computation in the Human Cortex, but Both Costs Are Needed to Predict Synapse Number. *Proceedings of the National Academy of Sciences* 118, 18 (May 2021), e2008173118. https://doi.org/10.1073/pnas.2008173118
[45] Samuel Liu, T Patrick Xiao, Can Cui, Jean Anne C Incorvia, Christopher H Bennett, and Matthew J Marinella. 2021. A domain wall-magnetic tunnel junction artificial synapse with notched geometry for accurate and efficient training of deep neural networks. *Applied Physics Letters* 118, 20 (2021), 202405.
[46] Danijela Marković, Alice Mizrahi, Damien Querlioz, and Julie Grollier. 2020. Physics for neuromorphic computing. *Nature Reviews Physics* 2, 9 (2020), 499–510.
[47] Neil McGlohon and Christopher D. Carothers. 2021. Toward Unbiased Deterministic Total Ordering of Parallel Simulations with Simultaneous Events. In *Proceedings of the Winter Simulation Conference* (Phoenix, Arizona) *(WSC '21)*. IEEE Press.
[48] Carver Mead. 1990. Neuromorphic Electronic Systems. *Proc. IEEE* 78, 10 (1990), 1629–1636.
[49] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.
[50] PK Muduli, T Higo, T Nishikawa, D Qu, H Isshiki, K Kondou, D Nishio-Hamane, S Nakatsuji, and YoshiChika Otani. 2019. Evaluation of spin diffusion length and spin Hall angle of the antiferromagnetic Weyl semimetal Mn 3 Sn. *Physical Review B* 99, 18 (2019), 184425.
[51] David M. Nicol. 1991. Performance Bounds on Parallel Self-Initiating Discrete-Event Simulations. *ACM Trans. Model. Comput. Simul.* 1, 1 (Jan. 1991), 24–50. https://doi.org/10.1145/102810.102812
[52] David M. Nicol. 1993. The cost of conservative synchronization in parallel discrete event simulations. *J. ACM* 40, 2 (April 1993), 304–333.
[53] Dmitri E. Nikonov and Ian A. Young. 2019. Benchmarking Delay and Energy of Neural Inference Circuits. *IEEE J. Explor. Solid-State Comput. Devices Circuits* 5, 2 (Dec. 2019), 75–84.
[54] Chenyun Pan and Azad Naeemi. 2016. Non-Boolean Computing Benchmarking for Beyond-CMOS Devices Based on Cellular Neural Network. *IEEE J. Explor. Solid-State Computat. Devices Circuits* 2 (2016), 36–43.
[55] Arun Parthasarathy, Egecan Cogulu, Andrew D Kent, and Shaloo Rakheja. 2021. Precessional spin-torque dynamics in biaxial antiferromagnets. *Phys Rev B* 103, 2 (2021), 024450.
[56] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia. 2012. The ROme OpTimistic Simulator: Core Internals and Programming Model. In *4th International ICST Conference on Simulation Tools and Techniques*.
[57] Kalyan S. Perumalla. 2014. *Introduction to Reversible Computing.* CRC Press/Taylor & Francis Group, Boca Raton, Florida.
[58] Adriano Pimpini. 2023. Towards Accessible Parallel Discrete Event Simulation of Spiking Neural Networks. In *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '23)*. Association for Computing Machinery, New York, NY, USA, 60–61. https://doi.org/10.1145/3573900.3593637
[59] Adriano Pimpini, Andrea Piccione, Bruno Ciciani, and Alessandro Pellegrini. 2022. Speculative Distributed Simulation of Very Large Spiking Neural Networks. In *Proceedings of the 2022 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '22)*. Association for Computing Machinery, New York, NY, USA, 93–104. https://doi.org/10.1145/3518997.3531027
[60] Mark Plagge, Christopher D. Carothers, and Elsa Gonsiorowski. 2016. NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*

*(SIGSIM-PADS '16)*. Association for Computing Machinery, New York, NY, USA, 233–244.

[61] Mark Plagge, Christopher D. Carothers, Elsa Gonsiorowski, and Neil Mcglohon. 2018. NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures. *ACM Trans. Model. Comput. Simul.* 28, 4, Article 30 (Sept. 2018), 25 pages.

[62] Peixin Qin, Zexin Feng, Xiaorong Zhou, Huixin Guo, Jinhua Wang, Han Yan, Xiaoning Wang, Hongyu Chen, Xin Zhang, Haojiang Wu, et al. 2020. Anomalous Hall effect, robust negative magnetoresistance, and memory devices based on a noncollinear antiferromagnetic metal. *ACS nano* 14, 5 (2020), 6242–6248.

[63] Maheshwar Pd. Sah, Hyongsuk Kim, and Leon O. Chua. 2014. Brains Are Made of Memristors. *IEEE Circuits and Systems Magazine* 14, 1 (2014), 12–36. https://doi.org/10.1109/MCAS.2013.2296414

[64] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. 2017. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv:1705.06963 [cs]* (May 2017). arXiv:1705.06963 [cs]

[65] Abhronil Sengupta, Aparajita Banerjee, and Kaushik Roy. 2016. Hybrid Spintronic-CMOS Spiking Neural Network with On-Chip Learning: Devices, Circuits, and Systems. *Phys. Rev. Applied* 6 (Dec 2016), 064003. Issue 6.

[66] William Severa, Craig M. Vineyard, Ryan Dellana, Stephen J. Verzi, and James B. Aimone. 2019. Whetstone: A Method for Training Deep Artificial Neural Networks for Binary Communication. *Nat Mach Intell* 1, 2 (Feb. 2019), 86–94.

[67] Ankit Shukla and Shaloo Rakheja. 2022. Spin-Torque-Driven Terahertz Auto-Oscillations in Noncollinear Coplanar Antiferromagnets. *Phys. Rev. Applied* 17, 3 (2022), 034037.

[68] Hans Skarsvåg, Cecilia Holmqvist, and Arne Brataas. 2015. Spin superfluidity and long-range transport in thin-film ferromagnets. *Physical review letters* 115, 23 (2015), 237201.

[69] Jacob M. Springer and Garrett T. Kenyon. 2020. It's Hard for Neural Networks To Learn the Game of Life. *arXiv:2009.01398 [cs, stat]* (Sept. 2020).

[70] Sudhir Srinivasan and Paul F. Reynolds. 1995. NPSI Adaptive Synchronization Algorithms for PDES. In *Proceedings of the 27th Conference on Winter Simulation* (Arlington, Virginia, USA) *(WSC '95)*. IEEE Computer Society, USA, 658–665. https://doi.org/10.1145/224401.224705

[71] Sudhir Srinivasan and Paul F. Reynolds. 1998. Elastic Time. *ACM Trans. Model. Comput. Simul.* 8, 2 (April 1998), 103–139. https://doi.org/10.1145/280265.280267

[72] Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi (Viji) Srinivasan, and Kailash Gopalakrishnan. 2020. Ultra-Low Precision 4-Bit Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1796–1807.

[73] Alexander N. Tait, Thomas Ferreira de Lima, Ellen Zhou, Allie X. Wu, Mitchell A. Nahmias, Bhavin J. Shastri, and Paul R. Prucnal. 2017. Neuromorphic Photonic Networks Using Silicon Photonic Weight Banks. *Sci Rep* 7, 1 (Aug. 2017), 7430. https://doi.org/10.1038/s41598-017-07754-z

[74] James M Taylor, Anastasios Markou, Edouard Lesne, Pranava Keerthi Sivakumar, Chen Luo, Florin Radu, Peter Werner, Claudia Felser, and Stuart SP Parkin. 2020. Anomalous and topological Hall effects in epitaxial thin films of the noncollinear antiferromagnet Mn 3 Sn. *Physical Review B* 101, 9 (2020), 094404.

[75] Jacob Torrejon, Mathieu Riou, Flavio Abreu Araujo, Sumito Tsunegi, Guru Khalsa, Damien Querlioz, Paolo Bortolotti, Vincent Cros, Kay Yakushiji, Akio Fukushima, et al. 2017. Neuromorphic computing with nanoscale spintronic oscillators. *Nature* 547, 7664 (2017), 428–431.

[76] Hanshen Tsai, Tomoya Higo, Kouta Kondou, Shoya Sakamoto, Ayuko Kobayashi, Takumi Matsuo, Shinji Miwa, Yoshichika Otani, and Satoru Nakatsuji. 2021. Large Hall Signal due to Electrical Switching of an Antiferromagnetic Weyl Semimetal State. *Small Science* 1, 5 (2021), 2000025. https://doi.org/10.1002/smsc.202000025

[77] Y. Y. Wang, C. Song, B. Cui, G. Y. Wang, F. Zeng, and F. Pan. 2012. Room-Temperature Perpendicular Exchange Coupling and Tunneling Anisotropic Magnetoresistance in an Antiferromagnet-Based Tunnel Junction. *Phys. Rev. Lett.* 109 (Sep 2012), 137201. Issue 13. https://doi.org/10.1103/PhysRevLett.109.137201

[78] Lloyd Watts. 1993. Event-Driven Simulation of Networks of Spiking Neurons. In *Advances in Neural Information Processing Systems*, Vol. 6. Morgan-Kaufmann.

[79] N. Wolfe, M. Plagge, C. D. Carothers, M. Mubarak, and R. B. Ross. 2018. Evaluating the Impact of Spiking Neural Network Traffic on Extreme-Scale Hybrid Systems. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 108–120.

[80] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]* (Sept. 2017).

[81] J Železnỳ, P Wadley, K Olejník, A Hoffmann, and H Ohno. 2018. Spin transport and spin torque in antiferromagnetic devices. *Nature Physics* 14, 3 (2018), 220–228.

[82] Mohammed A. Zidan, John Paul Strachan, and Wei D. Lu. 2018. The Future of Electronics Based on Memristive Systems. *Nat Electron* 1, 1 (Jan. 2018), 22–29. https://doi.org/10.1038/s41928-017-0006-8

Received 21 November 2022; revised 24 December 2023; accepted 4 February 2024