# NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures

MARK PLAGGE, CHRISTOPHER D. CAROTHERS, ELSA GONSIOROWSKI, and NEIL MCGLOHON, Rensselaer Polytechnic Institute

Neuromorphic computing is a broad category of non–von Neumann architectures that mimic biological nervous systems using hardware. Current research shows that this class of computing can execute data classification algorithms using only a tiny fraction of the power conventional CPUs require. This raises the larger research question: *How might neuromorphic computing be used to improve application performance, power consumption, and overall system reliability of future supercomputers?* To address this question, an open-source neuromorphic processor architecture simulator called *NeMo* is being developed. This effort will enable the design space exploration of potential heterogeneous compute systems that combine traditional CPUs, GPUs, and neuromorphic hardware. This article examines the design, implementation, and performance of *NeMo*. Demonstration of *NeMo*'s efficient execution using 2,048 nodes of an IBM Blue Gene/Q system, modeling 8,388,608 neuromorphic processing cores is reported. The peak performance of *NeMo* is just over ten billion events-per-second when operating at this scale.

CCS Concepts: • **Computing methodologies** → **Neural networks**; Discrete-event simulation; Massively parallel and high-performance simulations; • **Hardware** → **Emerging simulation**; *Emerging architectures*; *Neural systems*; • **Computer systems organization** → *Parallel architectures*;

Additional Key Words and Phrases: Neuromorphic architecture, massive parallel, discrete-event, time warp, reverse computation, biocomputing, neural net architecture, non von Neumann architecture

**ACM Reference format:**
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 (September 2018), 25 pages.
https://doi.org/10.1145/3186317

## 1 INTRODUCTION

In recent years, a new type of processor technology, called *neuromorphic computing*, has emerged. This new class of processor provides a brain-like computational model that enables complex neural network computations (e.g., data classification) to be done using significantly less power than von Neumann processors (Indiveri et al. 2011). For example, IBM has designed and created a neuromorphic processor, *TrueNorth* (Akopyan et al. 2015; Amir et al. 2013; Cassidy et al. 2013, 2014)

This work was supported by the Air Force Research Laboratory (AFRL), under award number FA8750-15-2-0078.
Authors' addresses: M. Plagge, C. D. Carothers, and N. McGlohon, Rensselaer Polytechnic Institute, Department of Computer Science, 110 8th Street, Troy, New York, 12180-3590; emails: plaggm@rpi.edu, chrisc@cs.rpi.edu, mcglon@rpi.edu; E. Gonsiorowski, Livermore Computing, L-556, Lawrence Livermore National Laboratory, Livermore, CA 95440; email: gonsie@llnl.gov.

1049-3301/2018/09-ART30 $15.00
https://doi.org/10.1145/3186317

that has 5.4 billion transistors arranged into 4,096 neurosynaptic cores with a total of 1 million spiking neurons and 256 million reconfigurable synapses. This architecture consumes only 65mW of power when executing a multi-object detection and classification program using real-time video input (30fps) for $400 \times 240$ pixel images. TrueNorth could run for over one week on a single charge inside today's smartphones. For a list of TrueNorth-capable algorithms and applications, see Esser et al. 2013.

This extremely low-power data analytics capability is particularly interesting as next generation High Performance Computing (HPC) systems are about to experience a radical shift in design and implementation. The current configuration of leadership class supercomputers provides much greater off-node parallelism than on-node parallelism. For example, the 20PF (petaFLOP) "Sequoia" Blue Gene/Q supercomputer located at Lawrence Livermore National Laboratory (LLNL) has over 98,000 compute nodes with each compute node providing at most 64 threads of execution. To reach exascale compute capabilities, a next generation system must be 50 times more power efficient. This dominating demand for power efficiency is resulting in future designs that dramatically decrease the number of compute nodes while increasing the computational power and number of processing cores. Case in point, a recent NASA vision report (Slotnick et al. 2014) predicts that exascale class supercomputers in the 2030 time frame will have only 20,000 compute nodes and the number of parallel processing streams per node will rise to nearly 16,000.

To meet the computational demands of these future designs, it has become a widely held view that on-node *accelerator* processors, in close coordination with multi-core CPUs, will play an important role in compute-node designs (Slotnick et al. 2014). These accelerators are currently used in two forms. The first are Graphical Processing Units (GPUs) that offer a single-instruction/multiple-data approach to parallelism, which matches the execution paradigm of graphics applications. GPUs offer a massive amount of numerical compute power at a very affordable price. The second form of compute-node accelerators is a mesh processor architecture such as the Intel Phi (Chrysos 2014). Here, a collection of lower clock-rate x86 cores are interconnected over an on-chip mesh network.

Given the advent of neuromorphic computing, future research will need to address how a neuromorphic processor can be used as an accelerator to improve the application performance, power consumption, and overall system reliability of future exascale systems. This system design is driven by the recent DOE SEAB report on HPC (Jackson et al. 2014). This report highlights the neuromorphic architecture as a key technology (especially in the next generation of supercomputing systems) for large-scale data processing.

To address this larger research question, an open-source processor architecture simulation framework is being developed as part of the *Super-Neuro* research project.[1] This effort will combine a number of modeling and simulation components to enable the design space exploration of potential hybrid CPU, GPU, and neuromorphic supercomputer systems. The key focus of this article is on the design, implementation, and performance of the neuromorphic architecture modeling component called *NeMo*. In particular, the key contributions of this article are:

—The design and implementation of an event-driven neuromorphic processor architecture model, *NeMo*, that is able to execute in parallel using optimistic event scheduling (Jefferson 1985) and reverse computation (Carothers et al. 1999).
—A discussion of two versions of this event-driven neuromorphic processor architecture model, one with fine-grained simulation components using explicitly defined instances of synapses (*NeMo-ES*) and one with a higher-performance Logical Process (LP)

[1]https://sites.google.com/site/superneuromorphic/.

implementation notably using a single "super-synapse" playing the role of all synapses for each core (*NeMo-SS*).

— An initial demonstration of *NeMo-ES*'s neuron model against the well-known Izhikevich model (Cassidy et al. 2013; Izhikevich 2003). The Izhikevich model exhibits well-known features of biological spiking neurons. In particular, phasic spiking and tonic bursting models are validated.
— A demonstration of *NeMo-SS*'s efficient execution on up to 2,048 Blue Gene/Q nodes for a 8,388,608 core neuromorphic processor model performing an "identity matrix" type neuron computation that generates a significant amount of neuron firing traffic. The peak performance of *NeMo-SS* is over ten billion events per second when operating at this scale.
— A weak scaling performance comparison between *NeMo-ES* and *NeMo-SS* on up to 1,024 Blue Gene/Q nodes, showing the increased performance of the new simulator design.

*NeMo* provides a framework to explore the neuromorphic hardware design space. While the work presented here simulated a specific neuromorphic hardware model, it is possible to extend and change the neuron, synapse, or axon models simulated. *NeMo* is able to show the behavior, accuracy, and activity of a neuromorphic hardware model. *NeMo* is able to scale to massively parallel execution, providing a tool that will enable the exploration of heterogeneous supercomputer designs containing neuromorphic hardware. Inspired by optimistic simulations of large-scale circuits in Gonsiorowski et al. (2012), *NeMo* implements optimistic simulation techniques.

The design, implementation, and integration of CPU, GPU, and network modeling components as part of the *Super-Neuro* project will be presented in other papers and is beyond the scope of the research presented here. The remainder of this article is organized as follows. Section 2 presents *NeMo*'s neuron model, which is derived from the model used in the TrueNorth processor (Cassidy et al. 2013), followed by the discrete-event implementation in Section 3. The validation and performance results are then presented in Sections 3.3 and 4. Last, related work and conclusions are presented in Sections 5 and 6, respectively.

## 2 BACKGROUND

Parallel Discrete-Event Simulations (PDES) consist of *Logical Process* (LP) objects, which communicate through *messages* or *events*. The LPs both encapsulate state and any computation within the simulation. However, for an LP to perform a computation or change its state, it must first be triggered by an event. Thus, changes throughout the simulation system occur via events flowing from one LP to another. When performing a parallel simulation, LP objects are placed on separate nodes connected across a network. While it is easy to ensure that events local to a node are in serial order, hiccups occur when events are arriving from the network.

PDES synchronization algorithms are used to keep simulation progress in sync across parallel nodes. Optimistic synchronization algorithms, such as Time Warp (Jefferson 1985), do not keep each of the parallel nodes in lock step but instead allow them to process events as they arrive. Nevertheless, there are still periods of global synchronization, called Global Virtual Time (GVT) calculation phases. The GVT calculations find the lowest timestamp on any unprocessed event. This allows the simulation system to reclaim memory from processed events.

Since there are no guarantees of global in-order execution of events, an LP may process a sequence of events out of serial order. To remedy this, the Time Warp algorithm keeps track of inter-event causality and requires LPs to have a "recovery mechanism" (Lin and Lazowska 1991).

One method of LP recovery is called *reverse computation* (Carothers et al. 1999). This method uses a function to "un-process" a given event. This allows LPs to reverse the effects of a series of events and begin forward event processing with a currently correct ordering.

When an LP detects an out-of-order event, the event is said to cause the LP to *rollback*. This rollback may require that certain messages be canceled. This cancellation process is done through anti-messages. These are messages with the sole purpose of undoing previously sent messages. Fujimoto 1999 defines two categories of rollbacks within Time Warp systems. The first, referred to as a primary rollback, is triggered by receiving a late message. These are triggered when a LP receives an event that was scheduled to arrive at a time less than the current simulated time. The second is referred to as a secondary rollback. These are triggered by an anti-message corresponding to a message that has already been processed by an LP.

## 2.1 ROSS

Rensselaer's Optimistic Simulation System, ROSS, is a prominent PDES engine (Carothers et al. 1999, 2000; Bauer et al. 2009; Holder and Carothers 2008). This ANSI C engine includes a reversible random number generator and is designed for fast and efficient performance. ROSS performs optimistic simulation using the Time Warp algorithm with reverse computation. ROSS also implements many other PDES scheduling algorithms, including the conservative YAWNS protocol (Nicol 1993; Nicol and Heidelberger 1996) and the real-time GVT protocol (Fujimoto and Hybinette 1997; Bauer et al. 2005). Overall, ROSS has been shown to be remarkably scalable (Barnes et al. 2013).

## 2.2 Neuromorphic Computing Models

*NeMo*'s neuromorphic processor architecture model is derived from the general neuron-synapse-axon behavior used in the IBM TrueNorth processor (Akopyan et al. 2015; Cassidy et al. 2014). The TrueNorth Leaky Integrate and Fire (TNLIF) model is further derived from the Leaky Integrate and Fire (LIF) model (Cassidy et al. 2013). We begin with an overview of hardware-based neuromorphic processor systems. We then present the LIF model, followed with details on the TNLIF model.

*NeMo* acts as a neuromorphic processor simulation model. It is designed not as a complete, cycle-accurate, hardware simulation, but as a generic neuromorphic hardware simulator that implements the TNLIF neuron model described. The *NeMo* model can simulate neuromorphic processors of arbitrary dimensions, allowing for novel processor performance benchmarking. *NeMo* also has the ability to add message processing inside the axons and synapses, potentially simulating more powerful or energy efficient neuromorphic processors, as described in Hasler and Marr (2013). This is in contrast to the *Compass* simulator, presented in Preissl et al. (2012), which is designed for spike accurate TrueNorth hardware simulations.

The TNLIF neuron model is a significantly enhanced version of the simple LIF model, described in detail in Izhikevich (2001). *NeMo* fully implements this neuron model. In Algorithm 1, the set of equations forming the TrueNorth neuron model is presented. Functions used in this model include signum:

$$sgn(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases},$$

a comparison function for stochastic operations:

$$F(s,p) = \begin{cases} 1, & |s| \geq p \\ 0, & |s| < p \end{cases},$$

and the single value Kroneker delta function:

$$\delta(x) = \begin{cases} 0, & \text{if } x \neq 0 \\ 1, & \text{if } x = 0 \end{cases}.$$

The TNLIF neuron model features a fully connected "neurosynaptic crossbar." This crossbar connects each input axon with all neurons. When an axon receives a spike, it sends signals to all connected synapses. The neuron integration equation is presented in Equation (1). At time $t$, if axon $i$ is active, then the synaptic activity, $A_i(t)$, is 1; otherwise, it is 0. In the equation, $w_{i,j}$ represents connectivity between axons and neurons. If $w_{i,j} = 1$, then there is a connection between axon $i$ and neuron $j$. If the value is 0, then there is no connection.

Each neuron assigns a type, represented by $G_i$, to each axon. Weights are then assigned to each axon type. $G_i$ is limited to four types, therefore each axon may be assigned one of four different weights by each neuron. Neuron weights are stored as signed integers, shown in the equation as $s_j^{G_i}$. The variable $b_j^{G_i}$ represents the integration mode: deterministic or stochastic. If the value is 0, then neurons update their membrane potential by taking the sum of each axon multiplied by each axon's weight: $\sum_{i=0}^{n-1}[s_j^{G_i}](A_i(t)w_{i,j})$.

Neurons can be configured to use stochastic synaptic and leak integration. Setting $b_j^{G_i} = 1$ enables stochastic synaptic integration and setting $c_j^{\lambda} = 1$ enables stochastic leak integration. Stochastic integration functions similarly for both leak and synaptic weight. For each integration event (either a synaptic weight or a leak computation), a random number is drawn and stored as $p_j$. If the drawn random number is higher than the relevant weight (synaptic weight $s_j^{G_i}$ or leak weight $\lambda_j$), then the neuron adds $sgn(\lambda)$ or $sgn(s_j^{G_i})$ to its membrane potential. Synapse integration is shown in Equation (1), and leak integration is shown in Equations (3) and (2).

The TNLIF neuron model enhances the leak functionality of the LIF model by adding positive or negative leak values, and a "leak-reversal" ability. Normal leak operation calculates the sign of the leak value, $\lambda_j$, stores this value as $\Omega$, and then integrates this value into the neuron's membrane potential. Leak sign calculation is shown in Equation (2), and integration is shown in Equation (3). Leak-reversal mode changes the behavior of the leak function such that if the neuron has a positive membrane potential, $\lambda_j$ is integrated directly, but if the neuron has a negative membrane potential, $-\lambda_j$ is integrated. In addition, if the membrane potential of a neuron is 0, then no leak is applied.

In addition to the deterministic threshold modes available, the TNLIF neuron model provides a stochastic threshold mode. To enable this mode, $M_j$ (a bitmask used to choose a random value) is set to a non-zero value. Then, $\eta_j$ is calculated every cycle by first generating a random number value, $p_j^T$, then taking the bitwise AND of $M_j$ and $p_j^T$, as seen in Equation (4). In Equations (7) and (5), $\eta_j$ is added to the threshold values before they are checked against the neuron membrane potentials.

The TNLIF model adds two new reset models to the LIF model. These modes are the normal reset mode that behaves similarly to the LIF model, a linear reset mode, and a non-reset mode. These values are chosen through the variable $\gamma_j$, and used in Equations (7) and (6). Linear reset mode subtracts the threshold value from the membrane potential. In non-reset mode, the membrane potential is not changed after a spike. These reset modes add additional functionality to the standard LIF neuron model.

TNLIF adds a negative threshold feature to the LIF. This negative threshold value is represented by $\beta_j$, an unsigned integer. This gives neurons the ability to have a membrane potential floor or a "bounce" feature. In the case of a floor setting, neurons with membrane potentials below $-\beta_j$ will set their values at $-\beta_j$. If the setting is set to a "bounce" value, then the neuron's membrane potential is reset to $-\beta_j$. The mode is set by changing the value of $\kappa_j$. Equation (7) shows the negative threshold check, and Equation (8) shows negative threshold reset and saturation.

The enhancements to the LIF model provided by TNLIF improves its flexibility and power. The additional stochastic integration and threshold features allow the TNLIF model to emulate continuous weight functions. Furthermore, the stochastic features allow neural networks trained with

**ALGORITHM 1:** TrueNorth leaky integrate and fire neuron model (TNLIF). $j$ refers to the $j$th neuron and $i$ refers to the $i$th input axon.

```
Integration:
    $V_j(t) = V_j(t-1) + \sum_{i=0}^{n-1} \left[ A_i(t)\, w_{i,j} \left[ \left(1 - b_j^{G_i}\right) s_j^{G_i} + b_j^{G_i}\, F\left(s_j^{G_i}, p_{i,j}\right) \mathrm{sgn}\left(s_j^{G_i}\right) \right] \right]$   (1)
Leak Integration:
    $\Omega = (1 - \epsilon_j) + \epsilon_j\, \mathrm{sgn}(V_j(t))$   (2)
    $V_j(t) = V_j(t) + \Omega \left[ \left(1 - c_j^{\lambda}\right) \lambda + c_j^{\lambda}\, F\left(\lambda_j, p_j^{\lambda}\right) \mathrm{sgn}\left(\lambda_j\right) \right]$   (3)
Threshold, Fire, Reset:
    $\eta_j = p_t^T \,\&\, M_j$   (4)
    if      $V_j(t) \geq \alpha + \eta_j$   (5)
        Spike
        $V_j(t) = \delta(\gamma_j) R_j + \delta(\gamma_j - 1)\,(V_j(t) - (\alpha + \eta_j)) + \delta(\gamma_j - 2)\, V_j(t)$   (6)
    elseif  $V_j(t) < -[\beta_j \kappa_j + (\beta_j + \eta_j)(1 - \kappa_j)]$   (7)
        $V_j(t) = -\beta_j\, \kappa_j + [-\delta(\gamma_j)\, R_j + \delta(\gamma_j - 1)\,(V_j(t) + (\beta_j + \eta_j)) + \delta(\gamma_j - 2)\, V_j(t)]\,(1 - \kappa_j)$   (8)
    endif
```

traditional back propagation techniques to run directly on the hardware (Esser et al. 2015). This neuron model's power and flexibility has been demonstrated in Cassidy et al. (2013) and was implemented on hardware in Akopyan et al. (2015).

The TNLIF model was originally developed through a software simulation tool called *Compass* (Preissl et al. 2012). *Compass* is a software tool provided by IBM to allow developers of neuromorphic software the ability to run code on a simulated TrueNorth processor. *Compass* is closed-source and proprietary, but there are some benchmark results available in Preissl et al. (2012) and Cassidy et al. (2014).

## 3 *NeMo* DISCRETE-EVENT IMPLEMENTATIONS

Based on the TNLIF model presented in the previous section, *NeMo* implements a neuromorphic architecture model using ROSS (Barnes et al. 2013; Bauer et al. 2009; Carothers et al. 2000). *NeMo* is a discrete event simulation based model of neuromorphic architecture. In this article, we look at two implementations within the discrete event simulation base: a simulation where every logical component is simulated as a distinct LP, and a super-synapse implementation where the synapse grid is implemented as a single LP. In this article, *NeMo* refers to both the explicit-synapse *NeMo-ES* and the "super-synapse" *NeMo-SS* versions.

*NeMo* simulates neuromorphic hardware using speculative simulation techniques. The properties of neuromorphic hardware are such that discrete event simulation may provide excellent performance. Spiking neural networks, as implemented in hardware, generally have a low rate of neuron activity at any given time. Furthermore, spikes do not carry more than a binary piece of information, making all spikes homogeneous across the network. This discrete output from the neurons, coupled with the low average network activity, produces a connection dense network with relatively low message activity. Based on promising results published by Lobb et al. (2005), *NeMo* implements a discrete-event simulation of neuromorphic hardware, using optimistic event scheduling.

Given the properties of neuromorphic hardware activity rates, we chose a parallel discrete event simulation that uses the Time Warp optimistic synchronization algorithm. Using an optimistic algorithm can lead to performance gains over a conservative time-stepped simulation when synchronization does not need to occur at every time-step. When simulating neuromorphic hardware, this speedup will be limited by how active the neurons are over time. The worst case scenario for this type of simulation would be a model where all neurons are actively sending spikes to each other. In this situation, the optimistic synchronization method will fall behind the performance of a conservative algorithm, due to the overhead induced by the optimistic synchronization algorithm.

The TNLIF model has specific limitations due to its implementation in hardware. *NeMo*, however, is not designed as a simulation of solely the TrueNorth processor hardware, rather it is a more generic neuromorphic processor simulation model. Given the constraints of the TNLIF model, *NeMo* implements all documented features of the hardware. In addition, *NeMo* supports significantly more features than the TrueNorth hardware. *NeMo* does not have the bit length constraints that are part of TrueNorth. *NeMo* may have a 64-bit signed integer value for weights, thresholds, and pseudo-random numbers. Furthermore, while *NeMo* operates with the same conceptual neurosynaptic crossbar that TrueNorth uses, the crossbar can be set to an arbitrary size, constrained only by memory of the system. This allows *NeMo* to simulate neurosynaptic cores of any size, across one or more MPI ranks. *NeMo* adds to these features by allowing the removal of the neurosynaptic crossbar completely, collapsing the model into a more traditional spiking neural network.

*NeMo* is also capable of simulating compute-on-synapse and compute-on-axon event models. This features give *NeMo* the ability to execute operations at the synapse or axon level, allowing for more complex neurosynaptic chip designs to be simulated.

*NeMo* partitions the model of a neuromorphic processor into individual components. By separating axon, synapse, and neuron objects into different LPs, *NeMo* is able to add processing features to the synapses and axons, a feature not found in current neuromorphic hardware. *NeMo-ES* simulates each axon, synapse, and neuron as individual LP types. This allows for flexibility in the simulation at the cost of increased numbers of LPs. In contrast, *NeMo-SS*'s implementation consolidates the synapse LPs in each core into a single synapse LP, providing a significant reduction in memory usage and event overhead.

The flexibility of *NeMo*'s simulation model allows for advanced axon → synapse → neuron connections to be modeled. A collection of axons, synapses, and neurons are grouped within a logical container, referred to as a synaptic core. *NeMo* can model thousands of neurosynaptic cores with each core containing hundreds of neurons and tens of thousands of logical synapses on one ore more physical processors. More details on *NeMo-ES* and *NeMo-SS*'s simulation performance is given in Section 4.

The remainder of this section discusses the implementations of *NeMo-ES* and *NeMo-SS*. This includes forward and reverse event functions for ROSS as well as a discussion on techniques used to prevent excessive message generation using a fanout technique, which maintains a stable message population. We first discuss *NeMo-ES*'s implementation details, followed by *NeMo-SS*'s enhancements and changes. We then briefly discuss the differences in implementation, simulation goals, and concepts between *NeMo* and the IBM TrueNorth hardware simulation software.

## 3.1 *NeMo-ES* Discrete-Event Implementation

For the benchmarking and testing of *NeMo*, we implement a model with similar capabilities as the TNLIF model. Therefore, we do not add any computation to the axon and synapse LPs. Table 1 shows the logical layout of neurons, axons, and synapses on a neurosynaptic core. When an axon

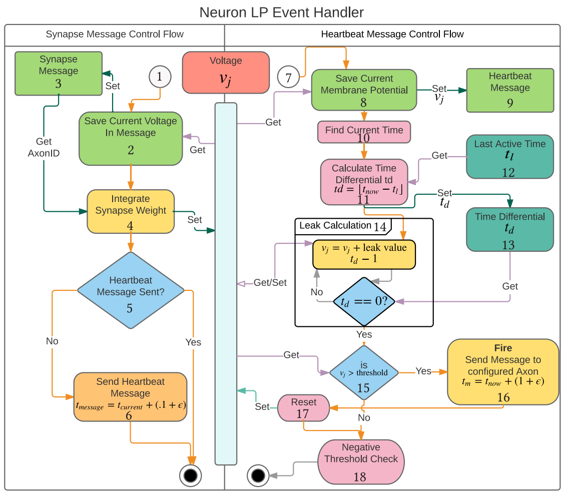

Neuron LP Event Handler
Synapse Message Control Flow
Heartbeat Message Control Flow
Synapse Message 3
Voltage $v_j$
Save Current Membrane Potential 8
Heartbeat Message 9
Save Current Voltage In Message 2
Find Current Time 10
Last Active Time $t_l$ 12
Calculate Time Differential td $td = \lfloor t_{now} - t_l \rfloor$ 11
Integrate Synapse Weight 4
Time Differential $t_d$ 13
Leak Calculation 14
$v_j = v_j +$ leak value $t_d - 1$
$t_d == 0?$
Heartbeat Message Sent? 5
is $v_j >$ threshold 15
Fire Send Message to configured Axon $t_m = t_{now} + (1+\epsilon)$ 16
Send Heartbeat Message $t_{message} = t_{current} + (.1+\epsilon)$ 6
Reset 17
Negative Threshold Check 18
Get AxonID
Get
Set
Get/Set
Yes
No

Fig. 1. The *NeMo-ES* neuron event flow. Details of each block, numbered 1 through 18, are discussed in Section 3.1.

Table 1. A Matrix Representation of a Neurosynaptic Core

| **Axons** | **Synapses** | | | |
|---|---|---|---|---|
| 0 | 0,0 | 0,1 | ... | 0, $n$ |
| 1 | 1,0 | 1,1 | ... | 1, $n$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $n$ | $n$, 1 | $n$, 2 | $n$, 3 | $n$, $n$ |
| **Neurons** | 0 | 1 | ... | $n$ |

receives a message it relays the message to each synapse in its row. In this model, the synapses simply relay any received message to the neuron in their column. Like the TNLIF model, there are no computations that occur when axons and synapses receive events; they simply relay their messages to the next element in the model.

In Figure 1, Blocks 1–18, we show the *NeMo* neuron model control flow for the forward event handler. The flow starts at the current simulation time, $t$, where $t$ is measured in microseconds. If $t > 1$, then there has been at least one neurosynaptic tick since the simulation has started. There are two event types that neurons receive: synapse messages and heartbeat messages. Synapse messages are set at a nanosecond resolution, with events occurring at $t + 0.0001 + \epsilon$, where $\epsilon$ is an extremely small random "jitter" value used to prevent ties in the scheduling of events to ensure a deterministic ordering of events.

$\epsilon$ is a small random value chosen to prevent event timestamp collisions. While the TimeWarp algorithm can theoretically return correct results when events occur simultaneously, the TimeWarp implementation in ROSS does not guarantee that multiple runs of the same model will produce the same output when event collisions occur. When ROSS manages rollback messages, these messages are organized in a priority queue. If two events have the same timestamp, then there is no way to guarantee what order they will appear in the queue of events. Despite this, *NeMo* may produce correct results even with event collisions, however, making *NeMo* deterministic and forcing *NeMo* to generate exactly reproducible output with every run required adding a random jitter factor to each event.

To wit, we use a random number generator that provides enough entropy to prevent event collision. Since the primary goal of *NeMo* is to simulate a neurosynaptic tick, we store the current tick value as the whole number in a 64 bit floating point number. Jitter is added as a small component of the floating point number. We assign 8 digits to the jitter value in the time variable, using the integrated ROSS random number generator. ROSS provides warnings that alert model developers to event collisions, allowing us to determine if any collisions have occurred. When combined with the incrementing values in the time-step, we found that this technique prevents event collisions during our tests.

Heartbeat messages are sent at a larger time-slice, $t + 0.1 + \epsilon$. In Figure 1, the synapse message processing is represented on the left column, and heartbeat messages are shown on the right.

The synapse message process begins in Block 1, when the neuron receives a synapse message. The neuron first saves the current voltage value, Block 2, a double precision floating point value $V_j$, in the synapse message, Block 3. This is to facilitate reverse computation, by saving $V_j$ in the message, when rolling back messages neurons are able to revert changes made during forward computation.

The neuron then performs the integration function, shown in Figure 1 as Block 4. This updates $V_j$ with a new value, computed by the integration function defined in Equation (1).

Neuron heartbeat messages are *NeMo*'s technique to synchronize neuron firing. In a LIF model, neurons integrate, leak, fire, and reset at specific intervals. We use heartbeat messages as way to ensure that neurons will perform the leak,fire, and reset functions only after receiving a spike event. This technique was chosen as a way to ensure that a neuron will leak and fire only after receiving a spike in a time-stepped method, while still providing performance gains that come from inactive neurons not computing at every time-step, as in a synchronized parallel model.

To increase performance, a heartbeat message is sent only when a neuron activates. In Block 5, the neuron checks if it has already sent a heartbeat message. If it has not, then it schedules a heartbeat message at $t + 0.1 + \epsilon$, in Block 6. This action completes the neuron's integration function for a particular axon. By executing this flow every time an axon message is received, *NeMo* re-creates the integration formula in Algorithm 1, Equation (1).

When a heartbeat message is received, as shown in Block 7, the neuron begins its leak, fire, and reset function. The neuron also saves its current membrane potential in the received message (Blocks 8 and 9).

The neuron then finds the current neurosynaptic time in Block 10. This is computed as $\lfloor t \rfloor$. In Block 11, the neuron calculates a time differential, $t_d$. This value represents how many neurosynaptic clock cycles have passed since this neuron has been active. By taking the last active time value, Block 12, and subtracting the current time, the neuron is able to determine how many times it needs to run the leak calculation, shown in Block 13. The neuron uses this time differential value to compute leak. By using a loop, the neuron is able to run the leak function, shown in Equations (2) and (3), $t_d$ times, bringing its voltage to where it would have been if the neuron had been calculating the leak function in a synchronous fashion. This loop is shown in Block 14.

Once the neuron has computed the leak function, it proceeds to check the positive threshold (Block 15), and either fires and resets or moves on to the negative threshold check. If $v_j$ is greater than the threshold, then the neuron will fire (Block 16) and reset (Block 17). A fire operation schedules a new message with ROSS at the next neurosynaptic clock time. Since the neurosynaptic clock operates at the integer scale, simply adding $1 + \varepsilon$ to the current time will schedule the fire event at the proper future time.

After the neuron completes the fire/reset functions, it then checks for negative threshold overflows (Block 18). If the neuron's voltage is beyond the negative threshold, then the neuron performs the negative threshold integration functions specified in Equation (8). With some neuron configurations, the reset voltage may be configured to subtract a value from the current membrane potential, rather than resetting the value to zero. As a final check after both reset functions have completed, if there is a remainder voltage that is past the threshold, then the neuron will schedule a heartbeat message for the next neurosynaptic tick.

The neuron has now completed one neurosynaptic tick. This example is that of a neuron as it receives spike events from connected axons. Here, the event flow assumes that the neuron is not self-firing. Certain configurations of neurons can create a situation where neurons are able to self-fire; setting the leak to a negative value, for example, will result in a neuron that spikes without receiving any axon inputs. In *NeMo*, this situation is handled by first detecting the presence of a self-firing neuron and then managing the neuron.

If a neuron has a positive leak (a leak that increases the membrane potential of the neuron) or a negative leak value with a corresponding negative reset value above the threshold value, then *NeMo* will detect this and apply a self-firing tag. This tag may be manually set when implementing neurons, allowing for other, non-detected self-firing neurons to be handled properly.

If the self-firing flag is set, then *NeMo* will detect this at the beginning of the simulation and schedule a heartbeat message for the next neurosynaptic tick. For all new ticks, this neuron will schedule heartbeat messages at each neurosynaptic tick to ensure that all self-firing events are simulated.

Reverse computation is handled through swapping states at key points in the neuron process and using bitfields to manage secondary state changes. The primary state change that occurs is $V_j$, the neuron's voltage. Neurons also contain a flag, marking when a neuron has sent itself a heartbeat message. When performing reverse computation, neurons must revert changes to both of these state elements.

Whenever a neuron receives a message from a synapse or receives a heartbeat message, before any changes are made to $V_j$, it saves the current current voltage in the incoming message. During reverse computation, neurons restore the saved voltage from the message. This reverts all integration, leak, and reset functions that changed $V_j$.

When a neuron receives a synapse message for the first time, it checks to see if it has sent a heartbeat message. If it has not, then it changes an internal flag, and sends the message. The neuron also changes the flag when receiving a heartbeat message. Neurons record boolean flag changes in a bitfield in the incoming message. If there is a non-zero entry in the bitfield during reverse computation, then the flag state is toggled.

Since *NeMo-ES* has individual LPs configured for each component, simulations have a large number of LPs running simultaneously. There are 2,164,260,864 LPs in our largest simulation experiment. If *NeMo-ES* sent messages at every time stamp, then it would send 66,048 messages per neurosynaptic core per tick. This large event population quickly becomes unmanageable due to memory constraints. To counter this, *NeMo-ES* implements a fanout technique for message transmission based on work done in LaPre et al. (2012).

In Figure 2, an example of the fanout message technique is shown. Here we see a neurosynaptic core with three axons, nine synapses, and three neurons. When a message is received by an axon, it sends an axon message to the first synapse in the neurosynaptic core at time $T + 0.0001 + \epsilon$. The synapse then sends two messages: first to the neuron attached to it, second to the next synapse in the row at $T + 0.0002 + \epsilon$. The next synapse does the same, until the final synapse has been reached. This technique generates far fewer messages, preventing memory usage issues.

## 3.2 *NeMo-SS* Discrete-Event Implementation

*NeMo-SS* builds on *NeMo-ES*, implementing all of the features of the original, while making some significant changes to the simulation design. The biggest change implemented is the introduction of a super-synapse, a single LP that manages axon $\rightarrow$ neuron communication. This design reduces the number of LPs by $n^2$, where $n$ is the number of neurons in a neurosynaptic core. For example, using the standard TrueNorth core size of 256 neurons, *NeMo-ES* creates 65,536 synapse LPs per core, while *NeMo-SS* creates 1 synapse LP per core. This reduction in the number of LPs reduces memory constraints and allows for larger simulations, as demonstrated in Section 4.

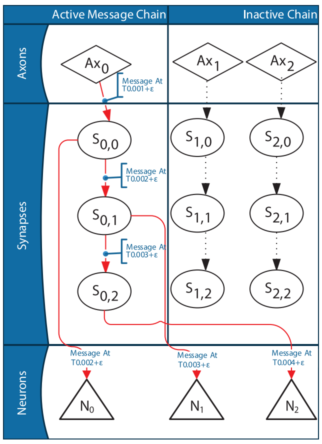
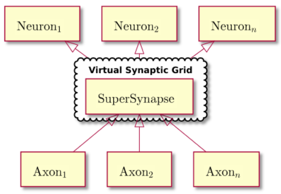
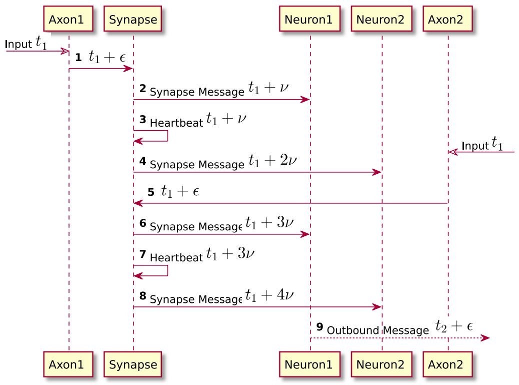


Fig. 2. Example event chain in *NeMo-SS* with three neurons per neurosynaptic core. In this diagram, an event is received at Axon 0 within a core at time $t$. At $t + 0.0001 + \epsilon$ Axon 0 sends a message to Synapse 0,0. Synapse 0,0 then sends a message at $t + 0.0002 + \epsilon$ to Neuron 0 and Synapse 0,1. Synapse 0,1 sends messages to Neuron 1 and Synapse 0,2 at $t + 0.0003 + \epsilon$. Synapse 0,2 then sends a message to Neuron 2 at $t + 0.0004 + \epsilon$. If no messages are received on Axon 1 and 2, then no messages are sent. Neurons will send outgoing spike messages, if applicable, at $t + 1.0 + \epsilon$.



(a) The *NeMo-SS* Core Layout Showing a Virtual Synapse Grid.



(b) *NeMo-SS* Event Implementation.

Fig. 3. Figure (a) shows the *NeMo-SS* core layout with the super synapse virtual synaptic grid. Figure (b) shows a trivial neurosynaptic core with two neurons. Input spikes arrive at the start of the current big tick, $t_1$ with a jitter value added. Jitter, represented by $\epsilon$, is added to every communication to prevent message collision. Some messages have both a counter and jitter applied to the time stamp, represented by $\nu$.

Further enhancements were made to the implementation of *NeMo-ES*, including optimizing of the in-memory representation of neurons and providing enhanced I/O options for simulation results.

The major design change in *NeMo-SS* versus *NeMo-ES* is the synapse grid design. *NeMo-SS* replaces individual synapse LPs with a single LP, here called a super-synapse. In Figure 3(a), *NeMo-SS*'s super-synapse layout is shown. The super-synapse represents the grid of synapse LPs used in *NeMo-ES*.

This super-synapse folds the synapse event fanout in *NeMo-SS* into a series of "heartbeat" messages. Compared with the event flow of *NeMo-ES*, shown in Figure 1, *NeMo-SS*'s event flow uses a heartbeat style message to manage events sent to the neurons. Figure 3(b) shows an example event chain in a trivial neurosynaptic core. For the purposes of illustration, Figure 3(b) shows a two neuron core receiving two outside messages.

To manage the "super-synapse's" heartbeat and neuron messages, we designed two time-steps: $\nu$, the internal clock time of a neurosynaptic core, and $t$, the simulated hardware time. As with *NeMo*'s fanout messages, $\epsilon$ is a small value added to scheduled event times that prevents collisions. These values are defined as

$t_n$ is the current simulated neurosynaptic hardware tick $n$. Each complete cycle of the hardware is considered one tick. Within ROSS, each event has a simulation time value, represented by a floating point number, associated with it. *NeMo-ES* and *NeMo-SS* use this to define the hardware time as

$$t_n = \lfloor t \rfloor.$$

$\epsilon$ represents a small "jitter" value. This is a similar value to the one described in the implementation of *NeMo-ES*.

$\nu$ is a small value that represents simulation steps that must be done within each hardware tick, $t_n$. This value is based on the number of neurons in each core. *NeMo-SS* calculates $\nu$ as

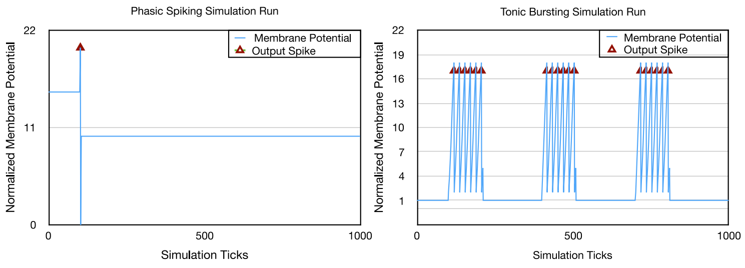
$$\nu = \frac{1}{2 * \text{Neurons Per Core}} + \epsilon.$$

Section 3.2 shows the event flow of a two neuron core inside *NeMo-SS*. For illustration, the core receives two spike events, both occurring at $t_1 + \epsilon$. We define our jitter value, $\epsilon$, such that $\epsilon \ll \nu$. Upon receiving the first axon message, the synapse LP sends a message to neuron 1 at time $t_1 + \nu$ and a heartbeat message at time $t_1 + \nu$. When the synapse receives this heartbeat message, it sends a message to neuron 2, scheduled for time $t_1 + 2\nu$. At this point, axon 2's message is received by the synapse, generating a new neuron message at $t_1 + 3\nu$ and a new heartbeat message at $t_1 + 3\nu$. When the synapse receives this second heartbeat message, a new neuron event is scheduled at $t_1 + 4\nu$. In this example, we assume that neuron 1 has a high enough voltage to spike, so it schedules an outbound neuron message at $t_2 + \epsilon$.

By separating the timestamps used by the synapse in this way, we observe that the order in which events are processed by the super-synapse do not affect the outcome of the simulation. Neuron integration can occur in any order in between the simulation ticks without changing the determinism of the model.

With *NeMo-SS*, we implemented a reverse computation technique for managing rollback events that affect neuron state. When a neuron in *NeMo-SS* receives a reverse message, the neuron applies a reverse integration function, along with a reverse leak function. These reverse computation methods were developed for both the LIF and TNLIF models, even though only the TNLIF model was used for benchmarking purposes. In general, spiking neuron integration, leak, and reset functions tend to be good candidates for reverse computation. Integration and leak functions are generally linear, and reset functions are almost always deterministic. *NeMo-SS* maintains the ability to perform incremental state swaps as well. Using state swapping will allow the quick addition of new models without devising reverse computation methods, as well as the addition of non-linear or other complex neuron functions.

An important feature of *NeMo* is the ability to not only simulate the TNLIF neuron model but virtually any neuromorphic model. This could be complex compute-on-synapse or compute-on-



(a) Izhikevich phasic spiking run. (b) Izhikevich tonic bursting run.

Fig. 4. Two Izhikevich Validation Run TNLIF Neuron Parameters and Results.

axon models where synapse and axon LPs act as more than just message carriers but also play parts on computation. It can also be as simple as a basic feed forward ANN.

To show this feature, the more simple Leaky Integrate and Fire (LIF) model described in Izhikevich (2004) was implemented to run on *NeMo*. The implementation of this model, while rudimentary, shows the capability of *NeMo*: that it is general and flexible.

The model is generally simpler than the TNLIF model in most respects but it has distinctive differences and flexibility. The TNLIF model's behavior is defined and restricted by hardware that the general LIF model (or any other conceivable model) is not bound to. For example, the data types that define the synaptic weights can be 64-bit signed floating point values instead of integers. In addition, there is no specified limit on the number of unique axon types that each neuron can store as potential inputs whereas the TNLIF model is limited to four types.

The significance of these differences is that *NeMo* is able to show scalable performance of different models ranging from simple to complex on arbitrary neuromorphic systems—including those that have not yet been implemented in hardware.

## 3.3 Accuracy Tests

Izhikevich implemented and reviewed 20 prominent features of biological neurons using a resonate-and-fire model (Izhikevich 2001). The TNLIF model was used to re-create many of these behaviors, demonstrating the utility and validity of the TNLIF model (Cassidy et al. 2013). *NeMo*, unlike Izhikevich's model and the *Compass* simulator used in the IBM reference paper, simulates TNLIF neurons using discrete events. Due to this difference, the discrete model can only approximate, although with a high degree of accuracy, Izhikevich's models. Neurons only update internal state when an input message is received or if they are a self-firing neuron (i.e., a neuron that can fire without first requiring input). However, we do re-create the neuron behavior observed in the TrueNorth neuron model. The result of this experimental run shows that while the inter-spike state of a simulated neuron may not be accurate, the spike times match what is observed when running these biological models using the Compass simulator.

To validate *NeMo*, we implemented two of the Izhikevich models Cassidy et al. implemented using the TNLIF model. Our goal was to match the behavior of these models, showing that *NeMo* correctly simulates the TNLIF model. To do this, we used the same parameters for each neuron as were used to generate the original results. A Phasic spiking neuron was configured this way, with a single axon input set to send spikes out every 200 ticks. The results of this run are shown in Figure 4(a).

We then implemented a tonic bursting neuron, again following the parameters used to re-create this behavior using the TNLIF model in COMPASS. In this configuration, we used two neurons and three axons. One axon was configured to send input spikes every 300 ticks. The neuron

Table 2. Neuron Validation Parameters

| Parameter | Neuron 0 Value |
|---|---|
| **Synaptic Weights $\left(s_j^{G_i}\right)$** | 0,20,0,0 |
| **Leak Value ($\lambda$)** | 2 |
| **Positive Threshold ($\alpha$)** | 2 |
| **Negative Threshold ($\beta$)** | −10 |
| **Reset Voltage ($R_j$)** | −15 |
| **Reset Mode** | Normal Negative Saturation |

(a) Phasic Spiking Parameters

| Neuron 0 Value | Neuron 1 Value |
|---|---|
| 1, −100, 0, 0 | 1, 0, 0, 0 |
| 1 | 0 |
| 18 | 6 |
| 20 | 0 |
| 1 | 0 |
| Normal Negative Saturation | Normal Negative Saturation |

(b) Tonic Bursting Neuron Parameters

parameters used for this run are shown in Table 2(b), and the membrane potential results are shown in Figure 4(b).

The information shown in Figure 4(a) and 4(b) visually presents neuron behavior that is nearly identical to the behavior observed using COMPASS. Slight differences in the values are a result of neurons updating state only when events warrant. We also do not record the membrane potential of the input axons. Despite this, we do see qualitatively similar neuron behaviors. Thus, the *NeMo* simulation model is able to re-create the simulation results produced by the IBM simulation tool, COMPASS.

## 4 EXPERIMENTAL PERFORMANCE

Understanding the performance of *NeMo-SS* within a massively parallel environment is important. The purpose of *NeMo-SS* is to allow for simulations of novel neuromorphic hardware and designs, for both exploration of novel neuromorphic hardware as well as simulation of neuromorphic hardware within a simulated heterogeneous HPC system. Providing the ability to simulate extremely large neuromorphic hardware networks will provide insights into massively connected hardware networks. To show that *NeMo-SS* will be able to simulate large structures of neuromorphic hardware, we ran *NeMo-SS* with extremely large networks in a massively parallel environment.

To accomplish this, we first examine a weak scaling experiment done on an IBM Blue Gene/Q, where we simulate up to 8,388,608 neurosynaptic cores with a total of 4,269,367,296 neurons. We then examine the strong scaling performance of a 65,536 neurosynaptic core simulation. We also compare these results with the first version of *NeMo*. A smaller run on an Intel based cluster is also examined, showing performance results that compare with the Blue Gene/Q architecture.

### 4.1 Experimental Setup

For each of the following experiments, we simulate TrueNorth-like neurosynaptic cores using the ROSS framework. In the weak scaling experiments, we compare a simulation containing neurosynaptic cores with 256 axon LPs, 1 synapse LP, and 256 neuron LPs, and a simulation containing neurosynaptic cores with 512 axon LPs, 1 synapse LP, and 512 neuron LPs. The largest *NeMo-SS* simulation contains a total of 8,598,323,200 LPs.

To test the performance of our model, we used a neurosynaptic core design that generates over 1,500 events per neurosynaptic core per tick. The neurons are configured such that they will fire a spike if they receive an input spike from an axon with the same ID. Each neurosynaptic core in this benchmark has weights such that an input from axon *i* will trigger neuron *i* to send one spike.

We have also implemented a "neuron connection pool" benchmark. In this network, every neuron is connected to 20 randomly selected axons (a pool of connections). All connected axons have

Table 3. Experimental Run Configurations

| Parameter | | | | | |
|---|---|---|---|---|---|
| Neurons Per Core | 256 | 256 | 256 | 512 | 512 |
| Synchronization Mode | GVT | Real-Time | GVT | Real-Time | GVT |
| NeMo Version | 1 | 2 | 2 | 2 | 2 |

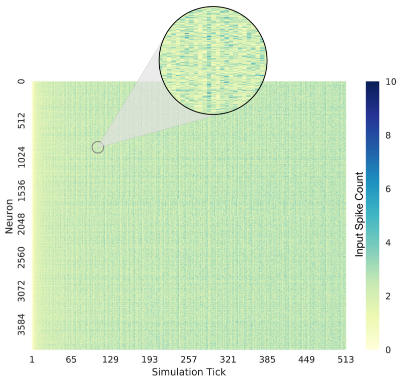


Fig. 5. Neuron activity in the identity matrix benchmark simulation. This chart represents the number of spikes sent by each neruon in a 4,096 neuron simulation across 512 ticks, demonstrating the workload generated by the benchmark model run. We observed an even distribution of neuron activity across the simulation running this benchmark.

a weight value of 1, and neurons have a threshold value of 5. As in the other benchmark model, neurons are connected to a pseudo-random output, with a probability of 0.9 of a connection to a different core.

The output destination of each neuron is set randomly with a 90% chance that it will output to a different neurosynaptic core. When the benchmark starts, each axon fires once. This benchmark setup generates an extremely large number of events, resulting in a larger workload than would be expected in a real-world application.

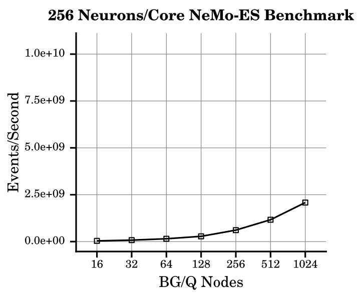
These simulations were performed on both an IBM Blue Gene/Q machine, and an Intel based cluster. Each node of the Blue Gene/Q features eighteen 1.6GHz processor cores, 16 of which are dedicated to application use (Haring et al. 2012). For the two remaining cores, one conducts operating system functions while the other series as spare. All nodes are connected by an effective, high-speed communication network (Chen et al. 2011).

The 16GB of DDR3 memory on each Blue Gene/Q node can be a limiting factor in memory intensive simulations. To allow for maximum utilization, each node is highly configurable in terms of parallelism. Each of the 16 processors can run up to 4 hardware threads (for a total of 64 MPI ranks per node) or the processor cores can be under-subscribed (with a minimum of 1 MPI rank per node). Our experiments test several parallel configurations.

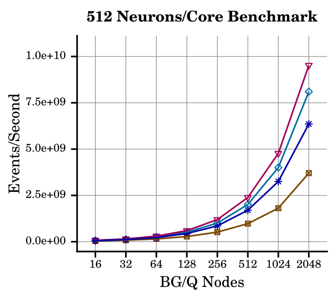
Each node of the Intel cluster the simulations were ran on consists of two four-core 3.3GHz Intel Xeon E5-2643 processors. The nodes are connected by a 56Gb FDR Infiniband network, providing high speed communication. Each node has 256GB of system memory available.

## 4.2 Blue Gene/Q Weak Scaling Experiments

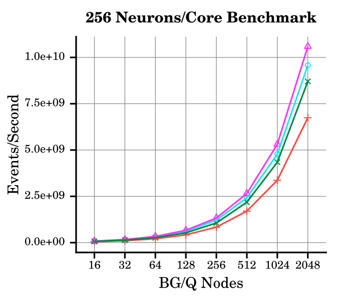
Our first set of experiments tested several configurations. For each *NeMo-SS* experiment, we configured the simulation with 16 neurosynaptic cores per MPI rank, for a total of 64 neurosynaptic



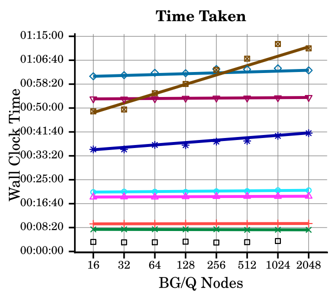
(a) *NeMo-ES* Weak Scaling 256 Neurons per Core



(b) *NeMo-SS* Weak Scaling 512 Neurons Per Core



(c) *NeMo-SS* Weak Scaling 256 Neurons Per Core



(d) *NeMo* Weak Scaling Wall Clock Times

256 Neurons/Core NeMo-SS
+ Neuron Pool Benchmark
× Neuron Pool Benchmark Realtime
○ Identity Matrix Benchmark
△ Identity Matrix Benchmark Realtime

512 Neurons/Core NeMo-SS
✳ Neuron Pool Benchmark Realtime
⊠ Neuron Pool Benchmark
◇ Identity Matrix Benchmark
▽ Identity Matrix Benchmark Realtime

NeMo-ES
□ Identity Matrix Benchmark

Fig. 6. Blue Gene/Q weak scaling performance experiments. Figure (a) shows the results of running *NeMo-ES* with 256 neurons per core with 16 cores per rank. Figure (b) shows the results of running *NeMo-SS* with 512 neurons per core with 64 cores per rank. Figure (c) shows the results of *NeMo-SS* with 256 neurons per core with 64 cores per rank. Figure (d) shows the wall clock time taken for each run.

cores per Blue Gene/Q node. In this weak scaling model, we ran *NeMo-SS* with 64 MPI ranks per Blue Gene/Q node. We ran two different neurosynaptic core configurations with *NeMo-SS*: one with 256 neurons per core, and the other with 512 neurons per core. The TrueNorth architecture has 256 neurons per neurosynaptic core. By adding a configuration with 512 neurons per core, we were able to simulate a theoretical hardware configuration. We ran the simulation for a total of 1,000 neurosynaptic core ticks. This is equivalent to running the TrueNorth hardware for 1s, as the hardware runs at 1,000Hz. The results of these runs are shown in Figure 6.

In Figure 6(a), we also show the results of a weak scaling experiment running *NeMo-ES*. In these runs, *NeMo-ES* was configured to simulate 256 neurons per neurosynaptic core, running the same benchmark configuration as *NeMo-SS*. Due to memory limitations, the *NeMo-ES* simulation was run with 16 neurosynaptic cores per MPI rank. This run was limited to 1,024 Blue Gene/Q nodes.

The *NeMo-ES* experiment achieved a peak performance of over 2 billion events per second when simulating 65,536 neurosynaptic cores on 1,024 Blue Gene/Q nodes with 64 MPI ranks per node. In Figure 6(d), *NeMo-ES* shows a near linear performance across the weak scaling

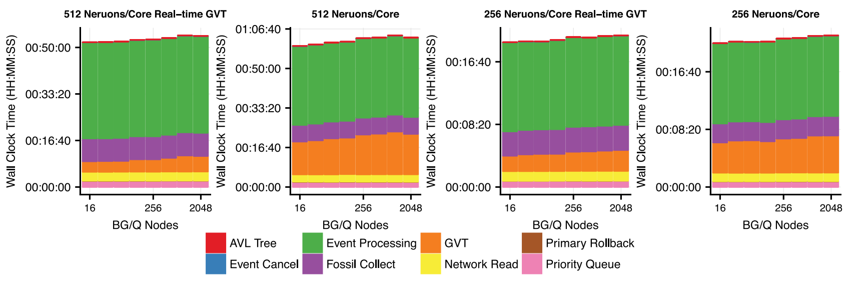


Fig. 7. *NeMo-SS* BG/Q weak scaling time detail.

simulation. Towards the upper end of the simulation, the wall-clock time increased. This is due to a slight load imbalance, caused by excessive GVT calculations. Every time a neuron fires, it has a 90% chance to send a signal to a neuron within a different neurosynaptic core. As the simulation encounters more network latency, the effects of rollbacks become more apparent in the simulation time. This long event chain (seen in Figure 2), coupled with the large number of random remote messages, results in an expected load imbalance. Generally, *NeMo-ES* shows near linear performance in weak scaling experiments.

For a more detailed analysis of *NeMo-ES*'s performance, see Plagge et al. (2016).

We ran *NeMo-SS* using two synchronization modes: GVT calculated based on the number of events processed and real-time GVT. In Figures 6(b) and 6(c), we show the performance of *NeMo-SS* using these two different synchronization modes. Interestingly, real-time GVT provides significantly better performance, especially when simulating 512 neurons per neurosynaptic core.

In Figure 6(d), the wall-clock time for these runs is shown. We found that both implementations of *NeMo* showed near linear wall-clock time across all identity matrix benchmark runs. *NeMo-SS*, running with 512 neurons per core, showed a slight increase in wall-clock time as the simulation size increased. This is likely due to the increased amount of time spent in GVT, as it can be seen in Figure 7.

The results of the neuron pool benchmark show the impact on *NeMo* of the significant increase in neuron activity within this benchmark network. The wall-clock time of the 512 neuron per core pool benchmark illustrates the potential limits of optimistic simulation in this scope. Switching from the standard GVT synchronization technique to the real-time technique improved scaling results as well.

Figure 7 shows the wall clock time spent in each phase of simulation during these runs. We observed that real-time GVT significantly reduced the amount of time spent processing GVT. *NeMo-SS* benefits greatly from the real-time GVT, showing reduced time spent waiting for synchronization.

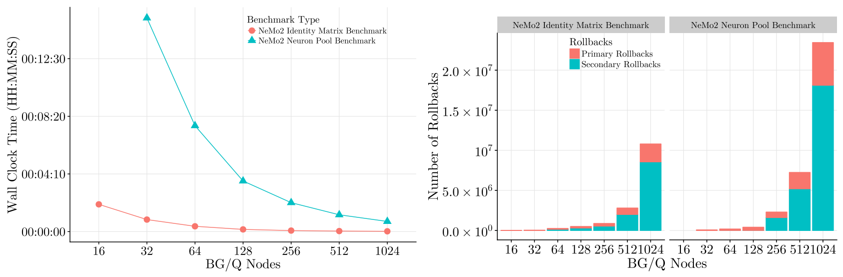
Optimistic synchronization in ROSS creates MPI barriers at each MPI rank after a specific number of events have been processed. Real-time synchronization creates these barriers after a set period of time. Real-time synchronization can provide performance increases over standard optimistic methods if many MPI ranks are waiting for a GVT to occur, while few MPI ranks are still processing events. This algorithm is based on the real time algorithm proposed by Fujimoto and Hybinette (1997), but adapted from a shared memory system to a fully distributed system.

In *NeMo*, we observed a significant burst effect of messages. Some MPI ranks will generate large numbers of messages quickly, while others are waiting to process new events. Given the burst-like nature of the simulation, the real-time synchronization protocol will generally provide better performance in this case. This performance increase is specific to the neurosynaptic model, as there are many bursts of high message activity followed by slower periods of message activity.

Table 4. Breakdown of Time Spent During the Simulations on 2048 Blue Gene/Q Nodes Each with 64 MPI Ranks

| | 512 Cores | 512 Cores Real-Time GVT | 256 Cores | 256 Cores Real-Time GVT |
|---|---|---|---|---|
| Priority Queue (enq/deq) | 130,615ms | 134,795ms | 49,065ms | 49,778ms |
| AVL Tree (insert/delete) | 1,316ms | 1,239ms | 764ms | 666ms |
| Event Processing | 2,005,299ms | 2,079,298ms | 697,964ms | 713,461ms |
| Event Cancel | 4,577ms | 2,808ms | 1,027ms | 1,002ms |
| GVT | 1,023,165ms | 333,997ms | 321,908ms | 167,047ms |
| Fossil Collect | 428,499ms | 497,998ms | 169,134ms | 200,164ms |
| Primary Rollback | 2,379ms | 2,308ms | 881ms | 45ms |
| Network Read | 187,937ms | 195,584ms | 74,874ms | 78,527ms |

The columns labeled with "Real-Time" text indicate the *NeMo-SS* runs that use the real-time GVT synchronization protocol, in contrast to the event count-based GVT method.



(a) Blue Gene/Q Strong Scaling Performance. (b) Blue Gene/Q Strong Scaling Rollbacks.

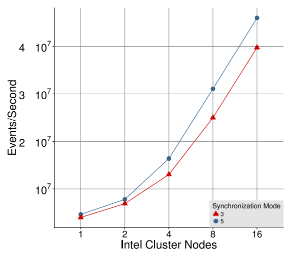
Fig. 8. *NeMo-SS* Blue Gene/Q strong scaling experiment results.

Figure 6(d) also shows the wall clock time taken for the *NeMo-SS* weak scaling simulation runs. For most of the *NeMo-SS* runs, a near linear wall-clock time is observed. The exception occurs when running the simulation with 512 neurons and standard GVT synchronization. This increase in execution time is most likely due to increased network communication overhead, along with the significantly higher time spent on event processing, as seen in Table 4.

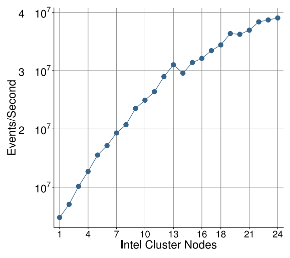
### 4.3 Blue Gene/Q Strong Scaling Experiments

To understand the ways in which the *NeMo-SS* model scales as parallelism increases, we ran a series of strong scaling experiments. Figure 8 shows performance results for a simulation of 65,536 neurosynaptic cores using 16 to 1,024 Blue Gene/Q nodes and the number of rollbacks observed. These experiments were run for 1,000 ticks resulting in over 9 billion net events. We achieved a peak performance of 5,834,092,242 events per second when we used 1,024 Blue Gene/Q nodes. This benchmark was run with the same randomly generated neuron model as the weak scaling experiments.

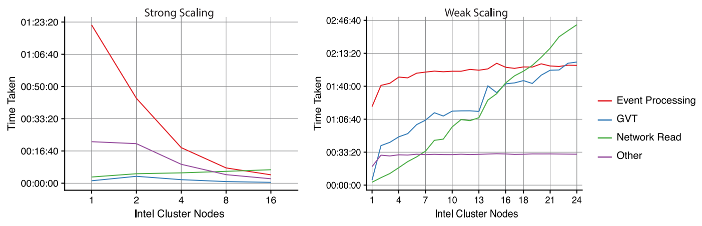
One thing to note is that *NeMo-ES* and *NeMo-SS* do not place a neurosynaptic core across multiple MPI ranks. This limits the maximum parallelization possible during strong scaling experiments. In this experiment, running 65,536 neurosynaptic cores gives a maximum of 65,534 MPI ranks. Figure 8(a) shows the simulation performance increase at a near linear rate until the number of neurosynaptic cores starts to equal the number of available MPI ranks. Eventually, the Blue Gene/Q's individual node compute power eclipses the available network bandwidth, slowing



(a) Strong Scaling Results



(b) Weak Scaling Results



(c) Intel Weak and Strong Scaling Time Details.

Fig. 9. *NeMo-SS* Intel cluster experiment results.

strong scaling performance gains. Figure 8(b) shows the increase in primary and secondary rollbacks as the communication overheads surpass the time spent during local event processing.

A major factor in the increase in overhead as the simulation reaches maximum parallelism is the time spent rolling back events. In Figure 8(b), the number of rollbacks dramatically increases as the number of nodes increases. When run at 1,024 Blue Gene/Q nodes, the number of rollbacks approaches the number of events computed.

The largest simulation done on the Blue Gene/Q used two racks, 2,048 nodes, to simulate 4,294,967,296 neurons in 8,388,608 neuromorphic cores. This simulation achieved a maximum event rate of 9,524,353,605 events per second when run with real-time synchronization. When the number of neurons per core was reduced to 256, the event rate increased to 10,589,662,119 events per second across 2,147,483,649 neurons.

## 4.4 Intel Cluster Experimental Results

Runs on the Intel cluster were done with an identical model configuration, but at a smaller scale. We ran both a weak scaling experiment as well as a strong scaling experiment on this cluster. The weak scaling experiment was configured with 512 neuromorphic cores per rank, each with 512 neurons per core. The neural network simulated was the same identity benchmark as was done in the Blue Gene/Q experiments. Each node of the Intel cluster ran with 8 MPI ranks, for a maximum of 192 ranks in the largest Intel cluster based experiment.

Figure 9(b) shows the results of the weak scaling experiment run on the Intel cluster.

In Figure 9(a), we show the results of a strong scaling experiment. This experiment was configured to simulate 4,096 neurosynaptic cores with 512 neurons per core. Each node in the cluster was configured with 16 MPI ranks, making the largest run 128 ranks. *NeMo-SS* initially shows excellent strong scaling performance, but as the size of the simulation increases the reduction in wall clock time reduces to sub-linear levels.

The sub-linear results using the Intel cluster are due to network bandwidth limitations compared with the individual node performance. The Intel cluster has a significantly slower network to compute power ratio than the Blue Gene/Q system. Each node of the Blue Gene/Q has a peak performance of 204.8 Giga FLOPS (GFLOPS) connected to a high speed network. In contrast, each node of the Intel cluster has two 4-core Intel Xeon E5-2643 processors running at 3.3GHz. Each processor has a peak theoretical performance of 105.6 GFLOPS for a total of 211.2 GFLOPS (Intel 2012). While each node of the Intel cluster has more power than the compute power of a Blue Gene/Q node, the network interconnect is slower than the 4-D torus implemented in the Blue Gene/Q system.

In Figure 9(c), the increase in time spent processing and waiting for the network is clearly visible. This chart shows the breakdown in wall-clock time taken for each component of the Intel cluster weak scaling experiments. As the time spent waiting on network communication increases, the amount of time spent handling GVT computations and waiting on GVT also increases. Compared to the Blue Gene/Q time breakdown shown in Figure 7, the difference in network performance of the systems becomes very apparent.

## 5 RELATED WORK

*NeMo* consists of a neuromorphic hardware simulation model, along with spiking neuron models. As indicated previously, the neurosynaptic core model of *NeMo* is based on the IBM TrueNorth chip, which has a "spike" accurate simulator called *Compass* (Cassidy et al. 2014).

A similar hardware specific neuromorphic system is the SpiNNaker Project (Furber et al. 2014). SpiNNaker is a specialized machine that is designed to optimally transmit a very large number of very small packets to enable models of how the brain performs communication operations as part of an overall neuron/brain modeling capability. Here, 40 byte packets are efficiently transmitted across to 1 million processing cores. The machine is organized into "nodes" similar to a Blue Gene/Q except that the core processing engine of each node is 18 ARM968 processor cores. Each ARM core has 96KB of local memory and 128MB of shared memory across all the processors. SpiNNaker reports being able to model on a single core several hundred point neurons performing calculations on par with Izhikevich's model with about 1,000 input synapses to each neuron. This fanout is about four times as big as currently supported in TrueNorth. However, the power consumed by SpiNNaker is much greater by several orders of magnitude. A 1,200 board system where each board support 48 nodes, which can model on the order of 10 to 20 million neurons consumes 75,000W of power whereas TrueNorth only consumes 65mW (or 0.065W) for 256,000 neurons.

Future neuromorphic hardware predictions where recently made by Hasler and Marr (Hasler and Marr 2013). Here, they present a road map for the construction of large-scale neuromorphic hardware systems. The metric used in this road map is called a MMAC, which is a unit of neuromorphic computation in the "millions" of neural multiple and accumulate operations. Hasler and Marr argue that if computation were done not only in the neurons but in the dendrites, which sit between the neuron cell (e.g., soma) and synapses, then it is possible to perform one million MAC operations per picowatt of power. This scale of computational power in equivalent to performing an exa-MAC or $2^{60}$ MAC operations per watt of power, which is on par with the computational power efficiency of the human brain.

In the computational neuroscience community, there are a number of spiking neuron simulators available that are using various modeling approaches to understand the biological function of neurons, dendrites, synapses, and axons. The most well known is NEURON (Brette et al. 2007), which is a simulation framework for creating and investigating empirically-based models of biological neurons and neural circuits. NEURON offers users the ability to select which numerical integration method to apply in solving the model equations. The default approach is an implicit Euler method. In Migliore et al. (2006), NEURON was extended to enable parallel neuron network simulations where each processor performs its own local equation integration over a subset of the neuron network. On the Blue Gene/P supercomputer it exhibited nearly linear speedup on 2,000 processing cores. Recently, Lin et al. (2015), constructed a multithreaded version of NEURON for reaction diffusion models that are implemented using a Time Warp with state-saving approach.

There has been work on simulating spiking neural networks using GPU acceleration as well. The "nemo" project, Fidjeland et al. (2009), uses GPU acceleration to simulate over 40,000 Izhikevich neurons in a biologically plausible network. The "nemo" project is designed to accelerate simulation of biologically accurate neural networks, whereas our *NeMo* project is tuned to simulate neuromorphic hardware design. GPU acceleration for simulation of spiking neural networks is promising (Carlson et al. 2014), and further work might be considered in simulating neuromorphic hardware using PDES techniques in tandem with GPU acceleration.

The Blue Brain Project[2] has attained world wide attention for its goals to construct high-fidelity, supercomputer-powered models of the human brain. This software is based on NEURON and uses the same numerical integration approaches. However, their brain models models can require very large data sets, because each neuron and synapse is distinct (Schürmann et al. 2014), which results in the cellular model for a human brain requiring 100PB of storage. This amount of data could be considered each and every time-step by the equation solvers in the Blue Brain simulator.

Finally, the only other optimistic neuron model with reverse computation is Lobb et al. (2005). In this 2005 PADS *Best Paper* work, a Hodgkin-Huxley (HH) neuron model is implemented, which demonstrates the performance viability of this approach. Speedups are demonstrated on an 8-node PIII cluster ranging from 1.5× to 3.5× for HH networks sizes of 25 to 400 neurons.

## 6 CONCLUSIONS & FUTURE WORK

We have presented *NeMo*, an open source[3] discrete event simulation model implemented using the ROSS simulation framework that allows for large-scale, flexible, simulation of neuromorphic processors. This simulation model allows for the creation of arbitrarily sized neuromorphic processors based on the TNLIF neuron model. It will also allow for experimentation with new neuromorphic processor designs and novel problem domains.

The results of this work show that discrete event simulation is a viable option for simulation of massive neuromorphic systems. Near linear scaling was achieved running *NeMo* on a Blue Gene/Q system with weak scaling. Our largest run of *NeMo* simulated 4,294,967,296 neurons contained in 8,388,608 neurosynaptic cores. This simulation generated over 9 billion events per second. When run with a similar neuron configuration as TrueNorth (256 neurons per neurosynaptic core), *NeMo* achieved over 10 billion events per second.

*NeMo* is also capable of simulating new configurations of neuromorphic hardware. The number of neurons per neurosynaptic core can be set to any value within the limits of 64 bit computer hardware. Furthermore, experiments can be done simulating neuromorphic processors that process messages upon receipt, allowing for "what-if" hardware designs. Since *NeMo* is built with the ROSS discrete event simulation framework, integration between *NeMo* and supercomputer simulation systems is possible. Combining the *NeMo* simulation model with a supercomputer design simulator will allow for experimentation with hybrid neuromorphic supercomputer designs.

One of the goals of *NeMo* is the ability to simulate different neuron models and hardware configurations. With this future goal in mind, *NeMo* has been designed to allow for the addition of other neuron models. The first model implemented is the TNLIF neuron model (Cassidy et al. 2013). However, *NeMo*'s neuron simulation is modular, allowing for new models to be "plugged-in" to the simulation. *NeMo* is capable of simulating any spiking neuron model, and is even capable

[2]Source: http://bluebrain.epfl.ch. Accessed on: Jan 4, 2016.
[3]Available At: https://github.com/markplagge/NeMo.

of having multiple neuron types per neurosynaptic core. The next steps for *NeMo* include the addition of Izhikevich's simple spiking neuron models, as defined in Izhikevich (2003) also.

We observed an increase in performance when using real-time GVT as opposed to an event based GVT. If an MPI rank running a neuromorphic core has to roll back messages, then the chain of messages can be long. For example, if a neuron output message has to be rolled back, then there is a potential for all of the neuron messages sent by the MPI rank's synapse to be rolled back. Using the real-time GVT reduced the time waiting for rollbacks, giving a significant performance increase.

Additionally, there is room for performance improvements in the design of *NeMo*. The super-synapse design used in *NeMo-SS* provides a significant reduction in the number of events generated per neurosynaptic operation when compared to the original *NeMo* implementation. Further enhancements could be made to this design. Currently, the synapse in a neurosynaptic core forwards messages to all neurons in the core regardless of the weights and connection in the destination neuron. If we sent neuron configuration to the synapse before the simulation started, then we could have a smarter synapse—one that would only send messages to neurons if the synaptic message will cause a change to the neuron's state.

When designing *NeMo*, we went with an optimistic event scheduler. While this has provided excellent results to date, we are interested in optimizing and testing a conservative scheduler. A conservative scheduler may outperform an optimistic scheduler depending on the neurosynaptic network configuration and activity. We would also like to examine the performance of a time-stepped simulation technique. While *NeMo* outperforms *COMPASS*, a time-stepped neuromorphic hardware simulation tool, we would like to see if a time-stepped discrete event simulation method would provide benefits.

The use of jitter as a technique to prevent event collision is another interesting area of future work. While we found no event collisions when using a random value, there may be situations where the entropy could be exhausted. We wish to investigate using a deterministic, iterative technique to guarantee no event collisions. We also wish to investigate the possibility of not using jitter at all in the simulation. As previously mentioned, the TimeWarp algorithm should be accurate even when events are scheduled at the same time. There may be a way to generate deterministic results within our simulation even without a jitter value.

Finally, the other major goal of *NeMo* is to present it as a stand-alone simulation framework. Eventually, *NeMo* should give users the ability to design and simulate custom neuromorphic hardware designs in an accessible way. We plan to add support for a high-level API, such as PyNN (Davison et al. 2009), or potentially a custom built API. We are also implementing other neuron model support in *NeMo*, with the intention of creating a versatile neuromorphic hardware design simulation tool. Further work includes the integration of *NeMo* and ROSS's new visualization tools (Ross et al. 2016) to provide more detailed views into the behavior of simulated neuromorphic hardware. Adding these features will be a major focus in the future work for this project.

## REFERENCES

F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. 2015. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 34, 10 (Oct 2015), 1537–1557. DOI: http://dx.doi.org/10.1109/TCAD.2015.2474396

A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha. 2013. Cognitive computing programming paradigm: A corelet Language for composing networks of neurosynaptic cores. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'13)*. 1–10. DOI: http://dx.doi.org/10.1109/IJCNN.2013.6707078

Peter D. Barnes, Jr., Christopher D. Carothers, David R. Jefferson, and Justin M. LaPre. 2013. Warp speed: Executing time warp on 1,966,080 cores. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS'13).* ACM, New York, NY, 327–336. DOI: http://dx.doi.org/10.1145/2486092.2486134

D. Bauer, G. Yaun, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. 2005. Seven-O'clock: A new distributed GVT algorithm using network atomic operations. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS'05).* 39–48. DOI: http://dx.doi.org/10.1109/PADS.2005.27

D. W. Bauer, C. D. Carothers, and A. Holder. 2009. Scalable time warp on blue gene supercomputers. In *Proceedings of the ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation.* 35–44. DOI: http://dx.doi.org/10.1109/PADS.2009.21

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 (2007), 349–398. DOI: http://dx.doi.org/10.1007/s10827-007-0038-6

K. D. Carlson, M. Beyeler, N. Dutt, and J. L. Krichmar. 2014. GPGPU accelerated simulation and parameter tuning for neuromorphic applications. In *Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC'14).* 570–577. DOI: http://dx.doi.org/10.1109/ASPDAC.2014.6742952

C. D. Carothers, D. Bauer, and S. Pearce. 2000. ROSS: A high-performance, low memory, modular time warp system. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation.* 53–60. DOI: http://dx.doi.org/10.1109/PADS.2000.847144

C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99).* 126–135. DOI: http://dx.doi.org/10.1109/PADS.1999.766169

A. S. Cassidy, R. Alvarez-Icaza, F. Akopyan, J. Sawada, J. V. Arthur, P. A. Merolla, P. Datta, M. G. Tallada, B. Taba, A. Andreopoulos, A. Amir, S. K. Esser, J. Kusnitz, R. Appuswamy, C. Haymes, B. Brezzo, R. Moussalli, R. Bellofatto, C. Baks, M. Mastro, K. Schleupen, C. E. Cox, K. Inoue, S. Millman, N. Imam, E. Mcquinn, Y. Y. Nakamura, I. Vo, C. Guok, D. Nguyen, S. Lekuch, S. Asaad, D. Friedman, B. L. Jackson, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. 2014. Real-time scalable cortical computing at 46 giga-synaptic OPS/watt with 100 speedup in time-to-solution and 100,000 reduction in energy-to-solution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14).* 27–38. DOI: http://dx.doi.org/10.1109/SC.2014.8

A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha. 2013. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'13).* 1–10. DOI: http://dx.doi.org/10.1109/IJCNN.2013.6707077

Dong Chen, Noel A. Eisley, Philip Heidelberger, Robert M. Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L. Satterfield, Burkhard Steinmacher-Burow, and Jeffrey J. Parker. 2011. The IBM blue gene/Q interconnection network and message unit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11).* ACM, New York, NY, Article 26, 10 pages. DOI: http://dx.doi.org/10.1145/2063384.2063419

George Chrysos. 2014. Intel Xeon Phi coprocessor-the architecture. *Intel Whitepaper.*

Andrew Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. 2009. PyNN: A common interface for neuronal network simulators. *Front. Neuroinformat.* 2 (2009), 11. DOI: http://dx.doi.org/10.3389/neuro.11.011.2008

S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. A. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha. 2013. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'13).* 1–10. DOI: http://dx.doi.org/10.1109/IJCNN.2013.6706746

Steve K. Esser, Rathinakumar Appuswamy, Paul Merolla, John V. Arthur, and Dharmendra S. Modha. 2015. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems 28,* C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 1117–1125. Retrieved from http://papers.nips.cc/paper/5862-backpropagation-for-energy-efficient-neuromorphic-computing.pdf.

A. K. Fidjeland, E. B. Roesch, M. P. Shanahan, and W. Luk. 2009. NeMo: A platform for neural modelling of spiking neurons using GPUs. In *Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors.* 137–144. DOI: http://dx.doi.org/10.1109/ASAP.2009.24

Richard M. Fujimoto. 1999. *Parallel and Distributed Simulation Systems* (1st ed.). John Wiley & Sons, Inc., New York, NY.

Richard M. Fujimoto and Maria Hybinette. 1997. Computing global virtual time in shared-memory multiprocessors. *ACM Trans. Model. Comput. Simul.* 7, 4 (Oct. 1997), 425–446.

S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. 2014. The spiNNaker project. *Proc. IEEE* 102, 5 (May 2014), 652–665. DOI: http://dx.doi.org/10.1109/JPROC.2014.2304638

Elsa Gonsiorowski, Christopher Carothers, and Carl Tropper. 2012. Modeling large scale circuits using massively parallel discrete-event simulation. In *Proceedings of the IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'12)*. IEEE Computer Society, Washington, DC, 127–133. DOI: http://dx.doi.org/10.1109/MASCOTS.2012.24

R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, G. Chiu, P. Boyle, N. Chist, and C. Kim. 2012. The IBM blue gene/Q compute chip. *IEEE Micro* 32, 2 (Mar. 2012), 48–60. DOI: http://dx.doi.org/10.1109/MM.2011.108

Jennifer Hasler and Harry Bo Marr. 2013. Finding a roadmap to achieve large neuromorphic hardware systems. *Front. Neurosci.* 7 (2013), 118. DOI: http://dx.doi.org/10.3389/fnins.2013.00118

Akintayo O. Holder and Christopher D. Carothers. 2008. Analysis of time warp on a 32,768 processor IBM blue gene/L supercomputer. In *Proceedings of the European Modeling and Simulation Symposium (EMSS'08)*. 284–292.

Giacomo Indiveri, Bernabe Linares-Barranco, Tara Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp HÃfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saãghi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. 2011. Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5 (2011), 73. DOI: http://dx.doi.org/10.3389/fnins.2011.00073

Intel. 2012. Intel Xeon Processor E5-2600 Series Specification White Paper.

Eugene M. Izhikevich. 2001. Resonate-and-fire neurons. *Neural Netw.* 14, 6–7 (2001), 883–894.

E. M. Izhikevich. 2003. Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 6 (Nov. 2003), 1569–1572. DOI: http://dx.doi.org/10.1109/TNN.2003.820440

E. M. Izhikevich. 2004. Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw.* 15, 5 (Sept. 2004), 1063–1070. DOI: http://dx.doi.org/10.1109/TNN.2004.832719

S. A. Jackson, M. McQuade, R. Shenoy, R. Giles S. Koonin, J. Hendler, P. Highnam, A. Jones, J. Kelly, C. Mundie, T. Ohki, D. Reed, K. Smith, and J. Tracy. 2014. *Report of the Task Force on High Performance Computing of the Secretary of Energy Advisory Board.* Technical Report. DOE.

David R. Jefferson. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7, 3 (July 1985), 404–425. DOI: http://dx.doi.org/10.1145/3916.3988

Justin M. LaPre, Christopher D. Carothers, Kenneth D. Renard, and Dale R. Shires. 2012. Ultra large-scale wireless network models using massively parallel discrete-event simulation. In *Defense and Military Modeling and Simulation 2012 (DMMS'12)*. Society for Computer Simulation International, San Diego, CA, 122–130.

Yi-Bing Lin and Edward D. Lazowska. 1991. A study of time warp rollback mechanisms. *ACM Trans. Model. Comput. Simul.* 1, 1 (Jan. 1991), 51–72.

Zhongwei Lin, Carl Tropper, Mohammand Nazrul Ishlam Patoary, Robert A. McDougal, William W. Lytton, and Michael L. Hines. 2015. NTW-MT: A multi-threaded simulator for reaction diffusion simulations in NEURON. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS'15)*. ACM, New York, NY, 157–167.

C. J. Lobb, Z. Chao, R. M. Fujimoto, and S. M. Potter. 2005. Parallel event-driven neural network simulations using the Hodgkin-Huxley neuron model. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*. 16–25. DOI: http://dx.doi.org/10.1109/PADS.2005.18

M Migliore, C. Cannia, W. W. Lytton, Henry Markram, and M. L. Hines. 2006. Parallel network simulations with NEURON. *J. Comput. Neurosci.* 21, 2 (2006), 119–129. arxiv:NIHMS150003

David Nicol and Philip Heidelberger. 1996. Parallel execution for serial simulators. *ACM Trans. Model. Comput. Simul.* 6, 3 (July 1996), 210–242.

David M. Nicol. 1993. The cost of conservative synchronization in parallel discrete event simulations. *J. ACM* 40, 2 (Apr. 1993), 304–333. DOI: http://dx.doi.org/10.1145/151261.151266

Mark Plagge, Christopher D. Carothers, and Elsa Gonsiorowski. 2016. NeMo: A massively parallel discrete-event simulation model for neuromorphic architectures. In *Proceedings of the Annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS'16)*. ACM, New York, NY, 233–244.

Robert Preissl, Theodore M. Wong, Pallab Datta, Myron Flickner, Raghavendra Singh, Steven K. Esser, William P. Risk, Horst D. Simon, and Dharmendra S. Modha. 2012. Compass: A scalable simulator for an architecture for cognitive computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE Computer Society Press, Los Alamitos, CA, Article 54, 11 pages. http://dl.acm.org/citation.cfm?id=2388996.2389070.

Caitlin Ross, Christopher D. Carothers, Misbah Mubarak, Philip Carns, Robert Ross, Jianping Kelvin Li, and Kwan-Liu Ma. 2016. Visual data-analytics of large-scale parallel discrete-event simulations. In *Proceedings of the 7th International*

*Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS'16).* IEEE Press, Piscataway, NJ, 87–97.

Felix Schürmann, Fabien Delalondre, Pramod S. Kumbhar, John Biddiscombe, Miguel Gila, Davide Tacchella, Alessandro Curioni, Bernard Metzler, Peter Morjan, Joachim Fenkes, Michele M. Franceschini, Robert S. Germain, Lars Schneidenbach, T. J. Christopher Ward, and Blake G. Fitch. 2014. *Rebasing I/O for Scientific Computing: Leveraging Storage Class Memory in an IBM BlueGene/Q Supercomputer.* Springer International Publishing, Cham, 331–347. DOI: http://dx.doi.org/10.1007/978-3-319-07518-1_21

J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis. 2014. *CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences.* Technical Report NASA/CR-2014-21878. NASA.

Received February 2017; revised November 2017; accepted February 2018