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

Mark Plagge
plaggm@rpi.edu

Christopher D. Carothers
chrisc@cs.rpi.edu

Elsa Gonsiorowski
gonsie@rpi.edu

Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York 12180-3590

ABSTRACT

Neuromorphic computing is a non-von Neumann architecture that mimics how the brain performs neural network types of computation in real hardware. It has been shown that this class of computing can execute data classification algorithms using only a tiny fraction of the power a conventional CPU would use to execute this algorithm. This raises the larger research question: *how might neuromorphic computing be used to improve the 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 hybrid CPU, GPU, and neuromorphic systems. The key focus of this paper is on the design, implementation and performance of *NeMo*. Demonstration of *NeMo*'s efficient execution on 1024 nodes of an IBM Blue Gene/Q system for a 65,536 neuromorphic processing core model is reported. The peak performance of *NeMo* is just over two billion events-per-second when operating at this scale.

Keywords

neuromorphic architecture; massive parallel; discrete-event; time warp; reverse computation; biocomputing; neural net architecture; non von Neumann architecture; neurosynaptic core

1. INTRODUCTION

In recent years, a new type of processor technology has emerged called *neuromorphic computing*. 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 [23]. For example, IBM has created an instance of the *TrueNorth* architecture [1, 2, 10, 11] 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 65 mW of power when executing a multi-object detection and classification program using real-time video input (30 fps) for 400×240 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. [15].

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 their design and implementation. The current configuration of leadership class supercomputers provides much greater off-node parallelism than on-node parallelism. For example, the 20 PFLOP "Sequoia" Blue Gene/Q supercomputer located at LLNL has over 98 thousand compute nodes with each compute node providing at most 64 threads of execution. In order 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 [42] 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 [42]. 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 [13]. 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 might a neuromorphic processor be used as an accelerator to improve the application performance, power consumption, and overall system reliability of future exascale systems. This systems design question is driven by the recent DOE SEAB report on HPC [27]. 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 de-

SIGSIM-PADS '16, May 15–18, 2016, Banff, Alberta, Canada.

ISBN 978-1-4503-3742-7/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2901378.2901392

veloped as part of the *Super-Neuro* research project (see: https://sites.google.com/site/superneuromorphic/). 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 paper is on the design, implementation and performance of the neuromorphic architecture modeling component called *NeMo*. In particular, the key contributions of this paper are:

- The design and implementation of an event-driven neuromorphic processor architecture model that is able to execute in parallel using optimistic event scheduling [28] and reverse computation [9].
- An initial validation of *NeMo*'s neuron model against the well known Izhikevich model [11, 24]. 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*'s efficient execution on up to 1024 Blue Gene/Q nodes for a 65,536 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* is over two billion events per second when operating at this scale.

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 paper is organized as follows. Section 2 presents *NeMo*'s neuron model which is derived from the model used in the TrueNorth processor [11] followed by the discrete-event implementation in Section 3. The validation and performance results are then presented in Sections 4 and 5. Last, related work and conclusions are presented in Sections 6 and 7, 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 perform any computations within the simulation. However, in order for an LP to perform a computation or change its state, it must be triggered by an event. Thus, changes throughout the simulation system occur via events flowing from one LP to another. When preforming 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 [28], 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" [30].

One method of LP recovery is called *reverse computation* [9]. 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 more correct ordering.

When an LP detects 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. Within Time Warp systems there are two categories of rollbacks [18]:

- **Primary Rollback**
  A rollback triggered by receiving a late message. For an LP at time t, a primary rollback occurs when it receives an event at a time less than t. This may cause some anti-messages to be sent.
- **Secondary Rollback**
  A rollback triggered by an anti-message corresponding to a message which has already processed by an LP.

2.1 ROSS

Rensselaer's Optimistic Simulation System, ROSS, is a prominent PDES engine [4, 7, 9, 22]. 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 implements many other PDES scheduling algorithms, including the conservative YAWNS protocol [36, 37]. ROSS has been shown to be remarkable scalable [3].

2.2 Neuromorphic Computing Models

NeMo's neuromorphic processor architecture model is derived from the general neuron-synaspe-axion behavior used in the IBM TrueNorth processor [1, 10]. The TrueNorth Leaky Integrate and Fire (TNLIF) model is further derived from the Leaky Integrate and Fire (LIF) model [11]. We begin with an overview of hardware-based neuromorphic processor systems. We then present the LIF model, followed with details on the TNLIF model.

2.2.1 Neuromorphic Hardware

Neuromorphic computing refers to hardware implementations of cognitive computing techniques. More specifically, the goal of neuromorphic computation is the design and development of neuron inspired hardware in an energy efficient package [33].

Neuromorphic hardware development has significantly progressed, giving rise to new processor designs [1, 33, 34, 38]. These hardware designs are based on spiking neural networks. Spiking neural networks have input, internal connections, and output components. Input elements are referred to as *axons*, output elements are called *neurons*, and the connectors between axons and neurons are called *synapses*. Signals sent from a neuron are generally referred to as "spikes," as they are treated as binary signals. These terms stem from more general Artifical Neural Network (ANN) models, which in turn are borrowed from neuroscience [40]. Neuromorphic processors operate on a synchronized clock, allowing them to receive, process, and send new messages in between external clock cycles. For example, the TrueNorth hardware architecture has an external clock rate of 1 kHz, allowing each neuron to receive and send a spike 1,000 times per second [1].

The current generation of neuromorphic hardware implements a spiking neuron model with binary outputs. Signals

Integration:

$$V_j(t) = V_j(t-1) \sum_{i=0}^{n-1} [x_i(t)\, s_i] \quad (1)$$

Leak:

$$V_j(t) = V_j(t) - \lambda_j \quad (2)$$

Threshold Check and Spike:

$$\textbf{if} \quad V_j(t) \geq \alpha_j \quad (3)$$

Spike

$$V_j(t) = R_j \quad (4)$$

$$\textbf{end if} \quad (5)$$

Figure 1: Leaky integrate and fire general neuron model

enter an axon, are passed to a synapse, then are processed by a neuron. The neurons in these models currently manage all of the computation—axons and synapses merely act as a signal transfer service to the neuron [21].

NeMo acts as a neuromorphic processor simulation model. It designed not as a complete, cycle-accurate, hardware simulation, but as a generic neuromorphic hardware simulator that implements the TNLIF neuron model described in [11].

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 [21]. This is in contrast to the *COMPASS* simulator, presented in [39], which is designed for spike accurate TrueNorth hardware simulations.

2.2.2 The LIF Neuron Model

The LIF model is a simple neuron model that is able to emulate some biological neuron functions [25]. Because it is so straightforward (and does not rely on partial differential equations), the LIF model is used in most neuromorphic hardware.

Neurons that implement the LIF model follow a simple pattern of execution, shown in Figure 1 [43]. Execution consists of an integration period, leak calculations, threshold checking, firing, and then reset. In Figure 1, the general form of the neuron equations are presented. During integration, shown in Equation (1), the neuron updates its internal voltage based on each synapse i's synaptic weight, s_i. This is calculated based on the synapse's activity at time t, shown as $x_i(t)$ in Equation (1). Next, the LIF neuron calculates leak, by subtracting the set leak value, λ, from the current membrane potential, shown in Equation (2). Next, in Equation (3), the neuron checks the threshold value, α, against the current membrane potential. If the current membrane potential is greater than α, the neuron spikes. If the neuron spikes, Equation (4) executes, setting the neuron membrane potential to the reset voltage, R_j. This model forms the basis of the TNLIF neuron model, and thus the basis of the *NeMo* simulation model.

2.2.3 The TrueNorth Neuron Model

The TNLIF neuron model is a significantly enhanced version of the simple LIF model. *NeMo* fully implements this neuron model. In Figure 2, the full 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 Kroneker delta function: $\delta(x)$.

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 (6). At time t, if an axon ,i, is active, 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 $w_{i,j}$ is 1, there is a connection between axon i and neuron j. If the value is 0, 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 $s_j^{G_i}$. $b_j^{G_i}$ sets deterministic or stochastic integration mode. If the value is 0, 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 λ_j), then the neuron adds $sgn(\lambda)$ or $sgn(s_j^{G_i})$ to its membrane potential. Synapse integration is shown in Equation (6), and leak integration is shown in Equations (7) and (8).

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 λ, stores this value as Ω, and then integrates this value into the neuron's membrane potential. Leak sign calculation is shown in Equation (7), and integration is shown in Equation (8). Leak-reversal mode changes the behavior of the leak function such that if the neuron has a positive membrane potential, λ is integrated directly, but if the neuron has a negative membrane potential, $-\lambda$ 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. Here, η_j is added to α_j and β_j. Then, η_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 (9). In Equations (10) and (12), η_j is added to the threshold values before they are checked against the neuron membrane potentials.

The TNLIF model also adds two new reset modes to the standard LIF model. TNLIF supports normal reset mode, a linear reset mode, and a non-reset mode. These values are

Integration:

$$V_j(t) = V_j(t-1) + \sum_{i=0}^{n-1} \Big[A_i(t)\, w_{i,j} \left[(1 - b_j^{G_i})\, s_j^{G_i} + b_j^{G_i}\, F(s_j^{G_i}, p_{i,j}) \operatorname{sgn}(s_j^{G_i}) \right] \Big] \tag{6}$$

Leak Integration:

$$\Omega = (1 - \epsilon_j) + \epsilon_j \operatorname{sgn}(V_j(t)) \tag{7}$$

$$V_j(t) = V_j(t) + \Omega \left[(1 - c_j^{\lambda})\, \lambda + c_j^{\lambda}\, F(\lambda_j, p_j^{\lambda}) \operatorname{sgn}(\lambda_j) \right] \tag{8}$$

Threshold, Fire, Reset:

$$eta_j = p_t^T \,\&\, M_j \tag{9}$$

if $V_j(t) \geq \alpha + \eta_j$ (10)

Spike

$$V_j(t) = R_j + \delta(\gamma_j - 1)\,(V_j(t) - (\alpha + \eta_j)) + \delta(\gamma_j - 2)\, V_j(t) \tag{11}$$

else if $V_j(t) < -[\beta_j \kappa_j + (\beta_j + \eta_j)(1 - \kappa_j)]$ (12)

$$V_j(t) = -\beta_j\, \kappa_j + [-\delta(\gamma_j)\, R_j + \delta(\gamma_j - 1)\,(V_j(t) + (\beta_j + \eta_j)) + \delta(\gamma_j - 2)\, V_j(t)]\,(1 - \kappa_j) \tag{13}$$

end if

Figure 2: TrueNorth leaky integrate and fire neuron model (TNLIF)

chosen through the variable γ_j, and used in Equations (11) and (12). Normal mode follows the standard LIF model. 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 β_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,, the neuron's membrane potential is reset to $-\beta_j$. The mode is set by changing the value of κ_j. Equation (12) shows the negative threshold check, and Equation (13) 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 traditional backpropagation techniques to run directly on the hardware [16]. Cassidy et al. demonstrated the flexibility and power of this neuron model in [11] and Akopyan et al. implemented this model in hardware in [1].

The TNLIF model was originally developed through a software simulation tool called *Compass* [39]. *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 benchmarks available in [10, 39]. Further discussion of *Compass* along with comparisons to *NeMo* can be found in Section 5.4.

3. NeMo DISCRETE-EVENT IMPLEMENTATION

Based on the TNLIF model presented in the previous section, the neuromorphic architecture model is realized as a discrete-event simulation using ROSS [3, 4, 8].

The TNLIF model has specific limitations due to its implementation in hardware. *NeMo*, however, is not designed as a simulation of the TrueNorth processor hardware, rather it is a more generic neuromorphic processor simulation model. With this set of design considerations, *NeMo* implements all of the features of the TrueNorth model. In addition, *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. This allows *NeMo* to simulate neurosynaptic cores of any size. For the purposes of our benchmark runs, we set the number of neurons to 256, the same that TrueNorth hardware implements.

NeMo is also capable of simulating compute-on-synapse event models. This feature gives *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. Each axon, synapse, and neuron are modeled as a unique Logical Process (LP) type. By having individual elements of the neuromorphic chip running as individual LP types, *NeMo* is able to add processing features to the synapses and axons. Furthermore, advanced axon $\rightarrow$ synapse $\rightarrow$ neuron connections could possibly be modeled. A collection of axons, synapses, and neurons are contained within a logical container, referred to as a neurosynaptic core. *NeMo* can model thousands of neurosynaptic cores with each core containing hundreds of neurons and axons and tens of thousands of synapses.

The remainder of this section discusses the implementation of *NeMo*. This includes the forward and reverse event functions for ROSS as well as a discussion on techniques used to prevent excessive message generation using a fanout technique to maintain a stable message population.

3.1 Forward Event Computation

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

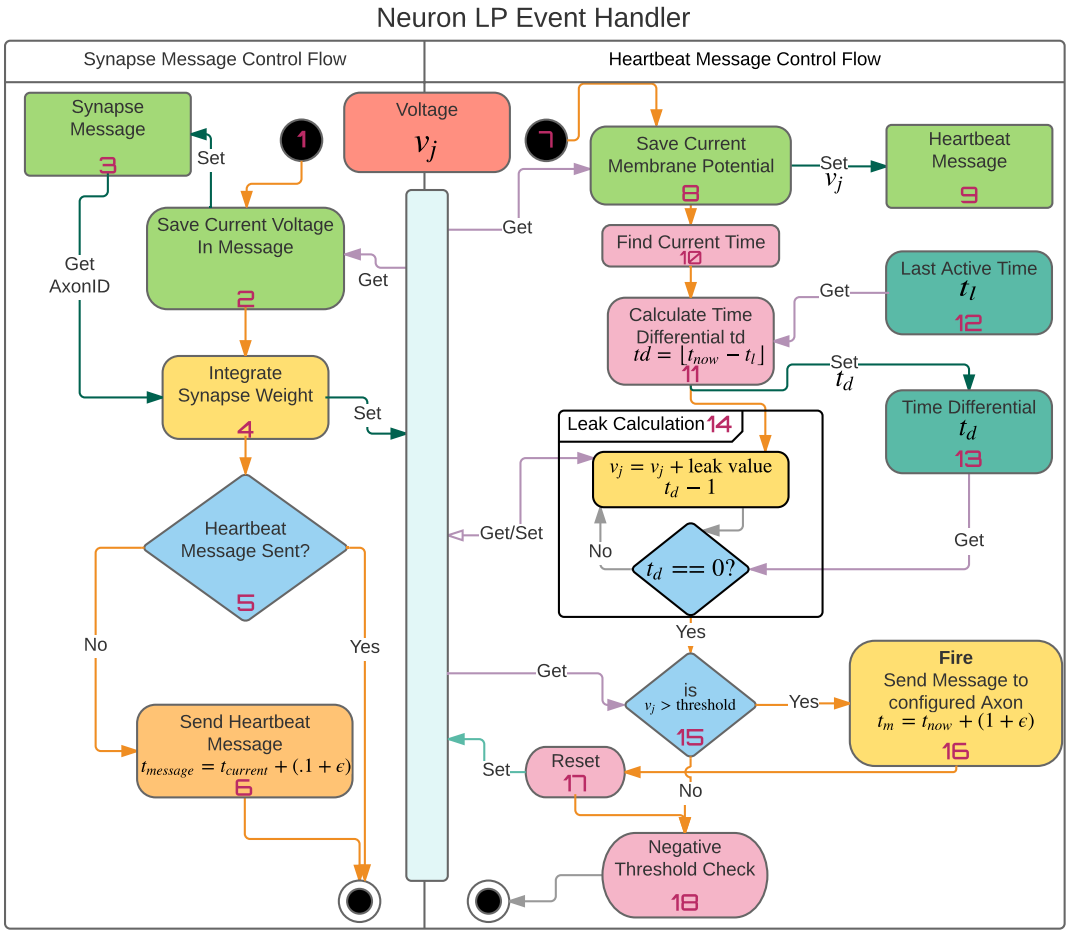

Figure 3: The *NeMo* neuron event flow. Details of each block, numbered 1 through 18, are discussed in Section 3.1.

synapse LPs. Section 3.1 shows the logical layout of neurons, axons, and synapses on a neurosynaptic core. When an axon 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 thier messages to the next element in the model.

In Figure 3, 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 $t > 1$, 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 $t + 0.0001 + \varepsilon$. Heartbeat messages are sent at a larger time-slice, $t + 0.1 + \varepsilon$. In Figure 3, the synapse message processing is represented on the left column, and heartbeat messages are shown on the right. We use ε to represent a "jitter" factor, an extremely small random value used to ensure a deterministic ordering of events.

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 3 as Block 4. This updates V_j with a new value, computed by the integration function defined in Equation (6).

Neuron heartbeat messages are *NeMo*'s technique to synchronize neuron firing. In an LIF model, neurons integrate, leak, fire, and reset at specific intervals. 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, it schedules a heartbeat message at $t + 0.1 + \varepsilon$, 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* recreates the integration formula in Figure 2, Equation (6).

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 (7) and (8), 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 fire and reset, or just move on to the negative threshold check. If $v_j >$ threshold, 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, the neuron performs the negative threshold integration functions specified in Equation (13).

The neuron has now completed one neurosynaptic tick.

3.2 Reverse Computation

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, 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, the flag state is toggled.

3.3 Fanout

Since *NeMo* 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* sent messages at every time stamp, 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,

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

Figure 4: A matrix representation of a neurosynaptic core.

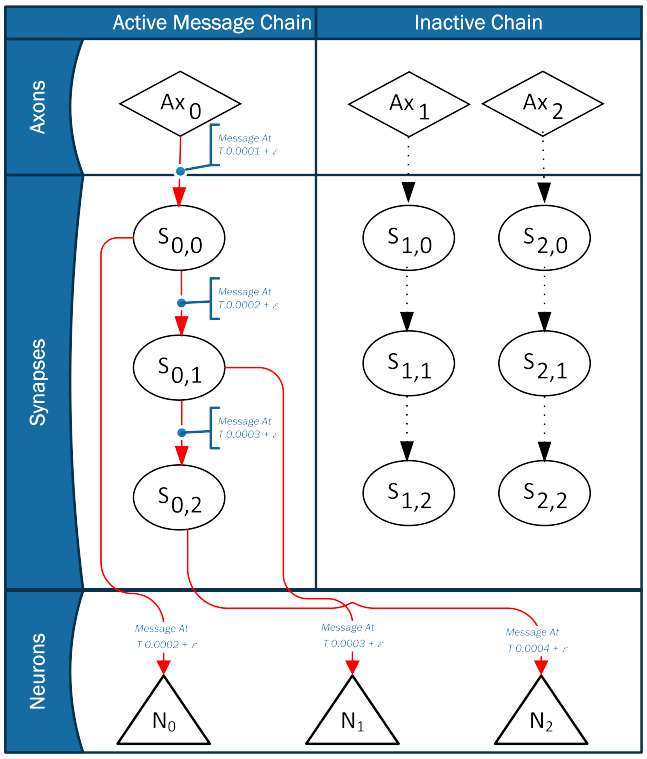

Figure 5: Example event chain in NeMo with 3 neurons per neurosynaptic core. In this diagram, an event is received at Axon 0 within a core at time t. At t+0.0001+ε Axon 0 sends a message to Synapse 0,0. Synapse 0,0 then sends a message at $t + 0.0002 + \varepsilon$ to Neuron 0 and Synapse 0,1. Synapse 0,1 sends messages to Neuron 1 and Synapse 0,2 at $t + 0.0003 + \varepsilon$. Synapse 0,2 then sends a message to Neuron 2 at $t + 0.0004 + \varepsilon$. If no messages are received on Axon 1 and 2, no messages are sent. Neurons will send outgoing spike messages, if applicable, at $t + 1.0 + \varepsilon$.

NeMo implements a fanout technique for message transmission based on work done in [29].

In Figure 5, 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 $T + 0.0001 + \varepsilon$. The synapse then sends two messages: first to the neuron attached to it, second to the next synapse in the row, at $T + 0.0002 + \varepsilon$. The next synapse does the same, until the final synapse has been reached. This technique generates far fewer messages, preventing memory usage issues.

4. VALIDATION

Izhikevich implemented and reviewed 20 prominent features of biological neurons using a resonate-and-fire model [26]. The TNLIF model was used to recreate many of these behaviors, demonstrating the utility and validity of the TNLIF model [11]. *NeMo*, unlike Izhikevich's model and the *Compass* simulator used in [11], simulates TNLIF neurons using discrete events. Due to this difference, it is impossible to exactly replicate Izhikevich's models. Neurons only update internal state when an input message is received or if they

are a self-firing neuron. However, we do recreate the neuron behavior observed by [11] in the TrueNorth neuron model.

To validate *NeMo*, we implemented two of the Izhikevich models Cassidey et al. implemented in [11]. Our goal was to match the behavior of these models, showing that *NeMo* correctly simulates the TNLIF model. To do this, we used similar parameters to the ones used in [11].Phasic spiking neurons were configured using the values shown in Table 1. A single axon was connected to this neuron that sent spikes out every 200 ticks. The results of this run are shown in Figure 6.

Table 1: Phasic Spiking Neuron Parameters.

Parameter	Neuron 0 Value
Synaptic Weights $\left(s_j^{G_i}\right)$	0,20,0,0
Leak Value (λ)	2
Positive Threshold (α)	2
Negative Threshold (β)	-10
Reset Voltage (R_j)	-15
Reset Mode	Normal Negative Saturation

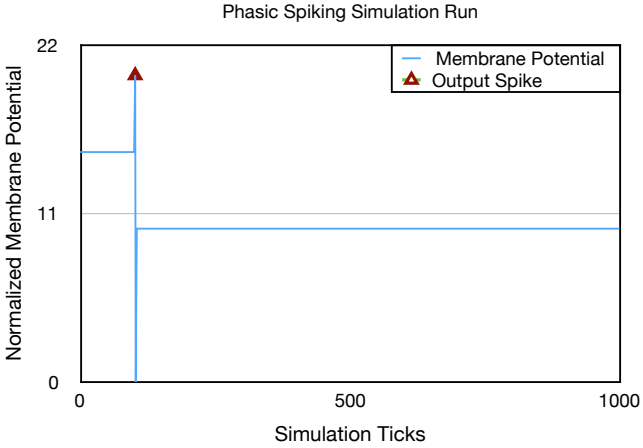

Figure 6: Izhikevich phasic spiking validation run.

We then implemented a tonic bursting neuron, again following the specifications set by [11]. In this configuration, we used two neurons and three axons. One axon was configured to send input spikes every 300 ticks. The neuron parameters used for this validation run are shown in Table 2, and the voltage results are shown in Figure 7.

The information shown in Figures 6 and 7 visually presents neuron behavior that is nearly identical to the behavior observed in [11]. 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 neruon behaviors. Thus, the *NeMo* simulation model is able to recreate the simulation results from [11].

5. EXPERIMENTAL PERFORMANCE

It is important to understand the performance of *NeMo* within massively parallel simulations. We first examine a weak scaling experiment, where we simulate up to 32,768

Table 2: Tonic Bursting Neuron Parameters.

Parameter	Neuron 0 Value	Neuron 1 Value
Synaptic Weights $\left(s_j^{G_i}\right)$	1, -100, 0, 0	1, 0, 0, 0
Leak Value (λ)	1	0
Positive Threshold (α)	18	6
Negative Threshold (β)	20	0
Reset Voltage (R_j)	1	0
Reset Mode	Normal Negative Saturation	Normal Negative Saturation

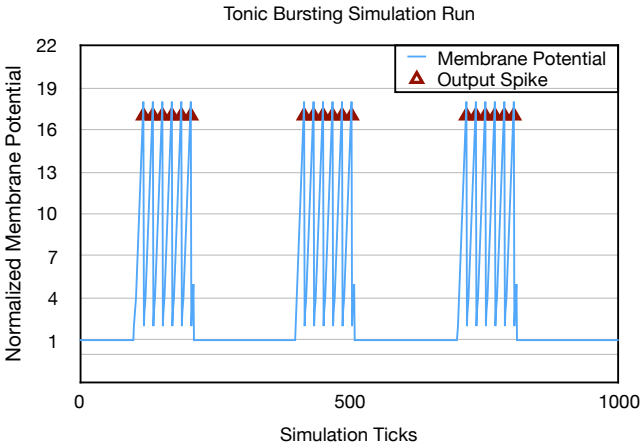

Figure 7: Izhikevich tonic bursting validation run.

neurosynaptic cores. Next we examine the strong scaling performance of a 8,192 neurosynaptic core simulation.

5.1 Experimental Setup

For each of the following experiments, we simulate TrueNorth-like neurosynaptic cores using the ROSS framework. Each neurosynaptic core connects 256 axon LPs, 65,536 synapse LPs, and 256 neuron LPs for a total of 66,048 LPs per core. We perform experiments with up to 32,768 neurosynaptic cores, giving a maximum number of 2,164,260,864 LPs in our largest simulation.

To fully test the performance of our model, we used a neurosynaptic core model which generates over 1,500 events per neurosynaptic core per tick. For this benchmark, each core consists of an "identity-matrix" of neurons. In this model, axon i will trigger synapse i, i, which triggers the neuron at i (see Section 3.1). The output destination of each neuron is set randomly with an 80% chance that it will output to a different neurosynaptic core. To start, each axon in the simulation fires. Overall, this creates an immense number of events, a larger workload than would be expected in a real-world application.

All simulations were performed on an IBM Blue Gene/Q machine. Each node of the Blue Gene/Q features eighteen 1.6 GHz processor cores, 16 of which are devoted to application use [20]. For the two remaining cores, one conducts operating system functionality while the other serves as a spare. All nodes are connected by an effective, high-speed communication network [12].

The 16 GB 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

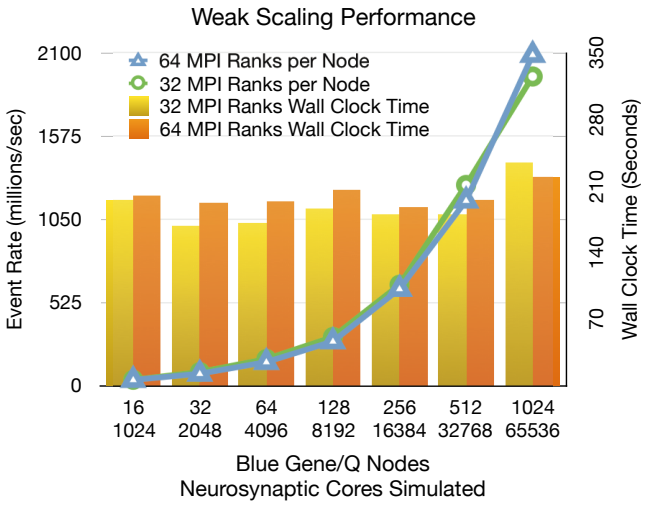

Figure 8: Weak scaling performance experiments.

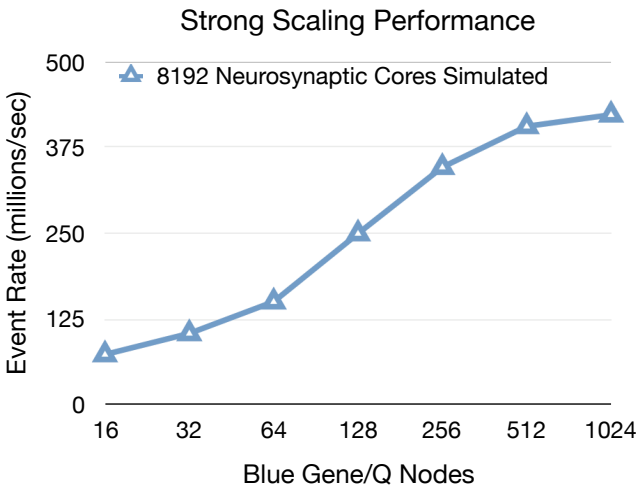

Figure 9: Strong scaling performance experiments.

Table 3: Breakdown of time spent during the simulation of 65,536 neurosynaptic cores on 1024 Blue Gene/Q nodes each with 64 MPI ranks.

Clock Cycle Category	Time Taken (seconds)	Percentage
Priority Queue (enq/deq)	1.8825	0.86%
AVL Tree (insert/delete)	0.0192	0.01%
Event Processing	38.2921	17.52%
Event Cancel	0.7486	0.34%
GVT	154.6155	70.75%
Fossil Collect	12.7742	5.85%
Primary Rollbacks	5.1278	2.35%
Network Read	5.0715	2.32%

(with a minimum of 1 MPI rank per node). Our experiments test several parallel configurations.

All experiments were performed using the time-warp based optimistic synchronization algorithm in ROSS.

5.2 Weak Scaling Experiment

Our first set of experiments tested two configurations: one and two neurosynaptic cores per MPI rank. These configurations ran on either 64 or 32 MPI ranks per Blue Gene/Q node, scaling from 16 to 1024 Blue Gene/Q nodes (see Figure 8). We achieved a peak performance of over 2 billion events per second when simulating 65,536 neurosynaptic cores on 1024 Blue Gene/nodes with 64 MPI ranks per node. These experiments simulated a total of 1,000 neurosynaptic core ticks.

Table 3 presents a breakdown of time spent during our peak performance simulation. These statistics are representative of all of our weak scaling experiments. The most noteworthy statistic is the time that the ROSS simulator spent performing GVT calculations. With less than 20% of simulation time being spent performing local event processing, we observe over 70% of the simulation time is spent performing GVT calculations. Since the GVT calculation is based around an `MPI_all_reduce` calculation, this indicates that there is a load imbalance within the simulation. That is, not all MPI ranks are reaching the blocking MPI reduction operation at the same time.

The slight load imbalance is to be expected. Every time a neuron fires, it has an 80% chance to send a signal to a neuron within a different synaptic core. Since the location of the receiver neuron is also chosen randomly, there is in an unpredictable, yet expected load imbalance across the simulation.

5.3 Strong Scaling Experiment

To understand the ways in which the *NeMo* model scales as parallelism increases, we ran a series of strong scaling experiments. Figure 9 shows performance results for a simulation of 8,192 neurosynaptic cores using 16 to 1,024 Blue Gene/Q nodes. These experiments were run for 1,000 ticks resulting in more than 13 billion net events. We achieved peak performance when we used 1,024 Blue Gene/Q nodes, where we observed over 421 million events per second. This benchmark was run with the same randomly generated neuron model as the weak scaling experements, with an 80% chance of neurons communicating to remote cores. One interesting thing to note is that *NeMo* does not place a neurosynaptic core across multiple MPI ranks. This is a limiting factor in the strong scaling results, as simulating 8,192 neurosynaptic cores gives a maximum of 8,192 MPI ranks. When running on 512 Blue Gene/Q nodes, there are 32,768 possible MPI ranks, and on 1,024 nodes there are 1,048,576 ranks available. We ran at these scales with 8,192 ranks, and the lack of increase in performance is attributable to this limitation in the *NeMo* system.

One interesting phenomena is observed when analyzing the running time and efficiency of the strong scaling experements, Figure 10. Here, we see that an unexpected correlation between overall simulation efficiency and the running time of the simulation: a *decrease* in efficiency corresponds to faster running times. This indicates a very low cost for performing an event rollback. Overall, the optimistic simulation is able to find more parallelism (and thus more speedup) despite incurring an increased number of rollback events.

Figure 11 shows a breakdown of the rollbacks observed during the strong scaling experiments. At first the number of rollbacks does increase as the parallelism increases however all experements on 128 nodes or more each incur approximately 7.7 million rollbacks. This indicates that our simulations have a maximum amount of parallelism where increases in hardware do not correlate to increases in performance.

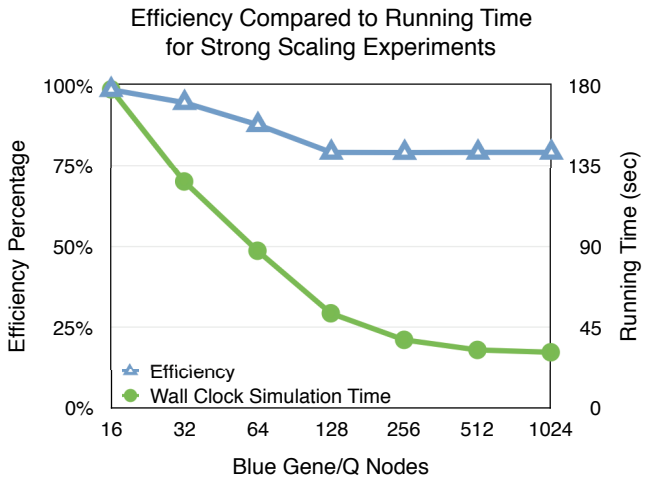

Figure 10: Comparison of the efficiency and the running time of the strong scaling experiments.

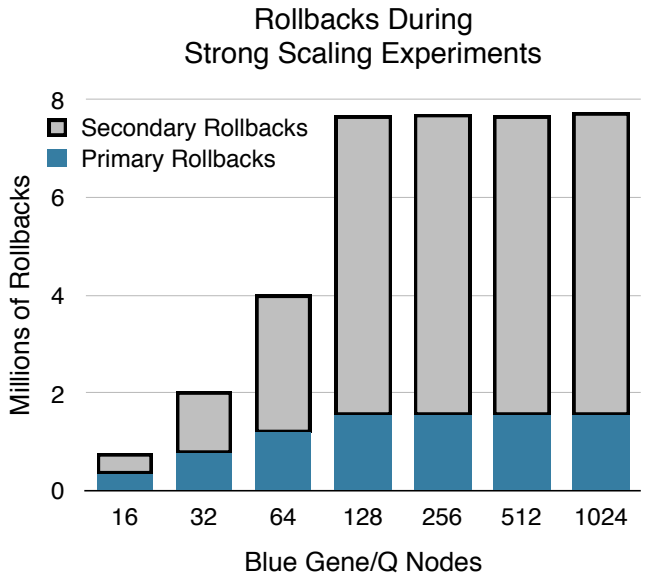

Figure 11: Breakdown of the primary and secondary rollbacks for the strong scaling simulation. Note that the net event population for these experiments is 13 billion events, but the maximum number of rollbacks observed is only 7.6 million.

Both Figure 10 and Figure 11 show that increasing the number of parallel nodes is only effective up to a point. At more than 128 Blue Gene/Q nodes, we see diminishing returns in performance scaling. In the 128 node experiment, each node simulates 64 neurosynaptic cores. We see that the overall workload is balanced (i.e., there is no decrease in performance which would indicate an over decomposition of the system). For the 128, 256, 512, and 1,024 node experiments, the communication overheads surpass the time spent doing local event processing. What is most intriguing about these experiments is that the number of rollbacks remains constant, despite an increase in parallelism.

5.4 Comparison with COMPASS

Comparing *NeMo* with IBM's own simulation software, *Compass*, poses some challenges. The intention of *NeMo* is to provide an open-source way to simulate various neuromorphic hardware designs. The *Compass* simulator is tuned for a similar purpose, but is tied into the TrueNorth architecture. Furthermore, *Compass* is proprietary software that we are unable to use on our benchmarking hardware. IBM Blue Gene/Q support for *Compass* was also eliminated in favor of focusing support on x86 architectures. These factors make a meaningful direct comparison impossible. However, in [39], benchmark runs of *Compass* are done using several models. Using these existing benchmarks, we can create a rough comparison between *NeMo* and *Compass*.

While both *NeMo* and *Compass* can exploit massively parallel supercomputer systems like the Blue Gene/Q, there are a number of key differences between the two. First, *Compass* employs a time-stepped algorithm which iterates over the set of neurons assigned to a particular thread (or MPI rank) and then iterates over each synaspe event that targets that neuron from the enclosing iteration loop. *NeMo* is implemented using a pure event-driven approach with all neuron, axon and synapse events being enqueued into a single priority queue. This approach avoids the cost of iterating over neurons which do not have any posted synaspe events.

In [39], *Compass* weak-scaling benchmarks are presented using the *CoCoMac* neuron model. We do not have access to the specific implementation details of the neuron model used in the paper, so we can not explicitly re-create the author's benchmarks in *NeMo*. CoCoMac is a message-sparse model, generating on average 1.3 spikes per simulated neurosynaptic core [39].

Given that we are not able to run the same model and we are not able to run *Compass* on our benchmark hardware, we have decided to compare the events per second produced by *Compass* with *NeMo*. To find this value for *Compass*, we took the number of spike events per simulated tick reported in [39] and made some reasonable assumptions about the underlying model. We took the values from the largest run of *Compass* that ran on 16 racks of IBM Blue Gene/Q that generated 22 million spikes per simulated tick. The paper does not specify if that value is for all spikes generated simulating the model, or if it was for remote-core spikes (spikes originating on a different node than the destination). To compare *NeMo*'s performance with *Compass*, we will assume the best-case scenario for *Compass*: that the 22 million spikes generated are only the remote spikes. The paper further specifies that the CoCoMac model simulated has some cores that generate a ratio of 80 remote spikes to 20 local spikes, and other cores that generate a 60/40 ratio. For the purposes of our comparison, we assume a best-case scenario for *Compass* and assume a 50/50 ratio of remote spikes versus local spikes. This assumption would mean that the 22 million spikes per simulated tick would be 50% of the total spikes generated by the simulation. For comparison, our benchmark model generated an average of 80% remote spikes per core.

The set of equations that approximate the performance of the *Compass* simulation are shown in Figure 12. To find the number of events per second, we assume that 1 axon and 256 synapse fan-out events, f are scheduled for each spike s event over the paper's reported 500 ticks t_{sim}, shown in Equation (14). We also multiply the total spikes reported, e_{total} by 2, per our assumption that 50% of the spikes are not remote. We then divide the calculated total number of events by the wall clock time taken by *Compass*, t_{wall} to get the spikes per second, e_{second}, shown in Equation (15). We then divide by the number of Blue Gene/Q racks used in

$$e_{\text{total}} = s_{\text{reported}} \times 2 \times f \quad (14)$$

$$e_{\text{second}} = s_{\text{total}}/t \quad (15)$$

$$e_{\text{second/rack}} = e_{\text{second}}/16 \quad (16)$$

Figure 12: Calculating *Compass*'s events per second per Blue Gene/Q Rack

the simulation r, to get *Compass*'s events per second per rack, shown in Equation (16). The complete equation used is shown in Figure 12.

These values were chosen to help represent the number of events *NeMo* produces. For every neuron spike event in *NeMo*, there are 256 neuron events generated with one axon event. Our largest current *NeMo* simulation ran on one rack of Blue Gene/Q, thus comparing *Compass*'s event rate per rack p er second will provide a roughly accurate gauge of performance.

This gives $((22\text{M} \times 2 \times 257) \times 500\,\text{ticks})/\,194\,\text{secs}/16\,\text{racks} = 1,821\text{M}\,\text{events}/\,\text{second}/\,\text{rack}$. Our benchmark runs of *NeMo* showed an event rate of $2,082\text{M}\,\text{events}/\,\text{rack}$. The weak scaling experements run on *NeMo* show 261M events per second more than *Compass*. While a direct comparison between *NeMo* and *Compass* is currently impossible, this result shows that *NeMo* is on par with the performance of *Compass*, and a viable option for simulation of neuromorphic hardware.

6. RELATED WORK

As indicated previously, the core neuron model of *NeMo* is based on the IBM TrueNorth chip which has a "spike" accurate simulator called *Compass* [10]. A detaled analysis of the differences between *Compass* and *NeMo* can be found in section 5.4.

A similar hardware specific neuromorphic system is the SpiNNaker Project [19]. 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 96 KB of local memory and 128 MB 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 fan-out 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 75,000 W of power whereas TrueNorth only consumes 65 mW (or 0.065 W) for 256 thousand neurons.

Future neuromorphic hardware predictions where recently made by Hasler and Marr [21]. 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 picawatt 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 [5] 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 [35], 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, Zhongwei et al. [31], constructed a multithread 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, [17], uses GPU acceleration to simulate over 40,000 Izhkavitch 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 [6], and further work might be considered in simulating neuromorphic hardware using PDES techniques in tandem with GPU acceleration.

The Blue Brain Project[1] 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 [41] which results in the cellular model for a human brain requiring 100 PB 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 by Lobb et al. [32]. 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.

7. CONCLUSIONS & FUTURE WORK

We have presented *NeMo*, an open source[2] 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. This simulation model will allow experimentation with new neuromorphic processor designs with new and novel problem domains.

The results of this work show that discrete event simulation is a viable option for simulation of massive neuromorphic

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

systems. Near linear scaling was achieved running *NeMo* on a Blue Gene/Q system with weak scaling. Our largest run of *NeMo* simulated 32,768 neurosynaptic cores, each containing 256 neurons, 65,536 synapses, and 256 axons, for 1,000 neurosynaptic ticks. The largest run simulated 8,338,608 neurons and axons with 2,147,483,648 synapses with an event rate of just over 1 billion events per second. Larger simulations are possible, along with different neurosynaptic core designs.

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 [11]. 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 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 [24].

Additionally, the design of *NeMo*'s message passing system does leave room for performance improvement. Significant simulation time is spent in GVT. Switching from ROSS's event based GVT to a real-time GVT algorithm potentially could improve execution speed. Further enhancements could be made to the way the neurosynaptic crossbar is implemented. Currently, *NeMo* forwards messages from synapses to neurons regardless if the neuron will act on the message. Adding connection information to the synapse would reduce message traffic, and potentially increase performance.

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 [14], or potentially a custom framework for describing neuromorphic hardware. We are also investigating including other neuron model support in *NeMo*, with the intention of creating a versatile neuromorphic hardware design simulation tool. Adding these features will be a major focus in the future work for this project.

8. ACKNOWLEDGMENTS

This work was supported by the Air Force Research Laboratory (AFRL), under award number FA8750-15-2-0078.

9. REFERENCES

[1] F. Akopyan, J. Sawada, et al. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[2] A. Amir, P. Datta, et al. Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores. In *IJCNN '13*, pages 1–10, Aug 2013.

[3] P. D. Barnes, Jr., C. D. Carothers, D. R. Jefferson, and J. M. LaPre. Warp speed: Executing time warp on 1,966,080 cores. In *ACM SIGSIM PADS 2013*, pages 327–336, Montreal, Canada, May 2013.

[4] D. Bauer, C. Carothers, and A. Holder. Scalable time warp on blue gene supercomputers. In *ACM/IEEE/SCS PADS '09*, pages 35–44, Lake Placid, NY, USA, 22–25 June 2009.

[5] R. Brette, M. Rudolph, et al. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, December 2007.

[6] K. D. Carlson, M. Beyeler, N. Dutt, and J. L. Krichmar. GPGPU accelerated simulation and parameter tuning for neuromorphic applications. In *ASP DAC '14*, pages 570–577. IEEE, 2014.

[7] C. Carothers, D. Bauer, and S. Pearce. Ross: A high-performance, low memory, modular time warp system. In *ACM SIGSIM PADS '00*, pages 53–60, Bologna, Italy, 28–31 May 2000.

[8] C. Carothers, D. Bauer, and S. Pearce. Ross: a high-performance, low memory, modular time warp system. In *ACM SIGSIM PADS '14*, pages 53–60, 2000.

[9] C. Carothers, K. Perumalla, and R. Fujimoto. Efficient Optimistic Parallel Simulations Using Reverse Computation. In *ACM SIGSIM PADS '99*, pages 126–135, Atlanta, GA, USA, 5 1999.

[10] A. S. Cassidy, R. Alvarez-Icaza, et al. Real-time scalable cortical computing at 46 giga-synaptic ops/watt with 100x speedup in time-to-solution and 100,000x reduction in energy-to-solution. In *ACM SC '14*, pages 27–38, Piscataway, NJ, USA, 2014. IEEE Press.

[11] A. S. Cassidy, P. Merolla, et al. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *IEEE IJCNN 2013*, 2013.

[12] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker. The IBM Blue Gene/Q Interconnection Network and Message Unit. In *ACM SC '11*, pages 1–10, Seattle, WA, USA, 12–18 Nov. 2011.

[13] G. Chrysos. Intel® Xeon Phi™ coprocessor-the architecture. *Intel Whitepaper*, 2014.

[14] A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. Pynn: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2(11):1–10, 27 Jan. 2009.

[15] S. Esser, A. Andreopoulos, et al. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *The 2013 Int. Joint Conference on Neural Networks*, pages 1–10, Aug 2013.

[16] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems 28*, NIPS '15, pages 1117–1125. Curran Associates, Inc., 2015.

[17] A. K. Fidjeland, E. B. Roesch, M. P. Shanahan, and W. Luk. NeMo: A platform for neural modelling of spiking neurons using gpus. In *IEEE ASAP 09.*, pages 137–144, July 2009.

[18] R. M. Fujimoto. *Parallel and Distributed Simulation*

Systems. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

[19] S. Furber, F. Galluppi, S. Temple, and L. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, May 2014.

[20] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, et al. The ibm blue gene/q compute chip. *Micro, IEEE*, 32(2):48–60, 2012.

[21] J. Hasler and H. B. Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in Neuroscience*, 7(118), 2013.

[22] A. O. Holder and C. D. Carothers. Analysis of Time Warp on a 32,768 Processor IBM Blue Gene/L Supercomputer. In *Proc. 20th Eur. Modeling and Simulation Symp.*, EMSS '08, pages 284–292, Amantea, Italy, 17–19 Sept. 2008.

[23] G. Indiveri, B. Linares-Barranco, et al. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5, 2011.

[24] E. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, Nov 2003.

[25] E. Izhikevich. Which model to use for cortical spiking neurons? *Neural Networks, IEEE Transactions on*, 15(5):1063–1070, Sept 2004.

[26] E. M. Izhikevich. Resonate-and-fire neurons. *Neural Networks*, 14(6–7):883 – 894, 2001.

[27] S. A. Jackson, M. McQuade, R. Shenoy, R. G. S. Koonin, J. Hendler, P. Highnam, A. Jones, J. Kelly, C. Mundie, T. Ohki, D. Reed, K. Smith, and J. Tracy. Report of the task force on high performance computing of the secretary of energy advisory board. Technical report, DOE, August 2014.

[28] D. R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, July 1985.

[29] J. M. LaPre, C. D. Carothers, K. D. Renard, and D. R. Shires. Ultra large-scale wireless network models using massively parallel discrete-event simulation. *Transactions of The Society for Modeling and Simulation Int.*, Oct. 2012.

[30] Y.-B. Lin and E. D. Lazowska. A study of time warp rollback mechanisms. *ACM Trans. Modeling and Comput. Simulation*, 1(1):51–72, Jan. 1991.

[31] Z. Lin, C. Tropper, M. N. Ishlam Patoary, R. A. McDougal, W. W. Lytton, and M. L. Hines. Ntw-mt: A multi-threaded simulator for reaction diffusion simulations in neuron. In *SIGSIM PADS '15*, pages 157–167, New York, NY, USA, 2015. ACM.

[32] C. J. Lobb, Z. Chao, R. M. Fujimoto, and S. M. Potter. Parallel event-driven neural network simulations using the hodgkin-huxley neuron model. In *Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on*, pages 16–25, June 2005.

[33] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *IEEE CICC '11 IEEE*, pages 1–4, Sept 2011.

[34] P. A. Merolla, J. V. Arthur, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[35] M. Migliore, C. Cannia, W. W. Lytton, H. Markram, and M. L. Hines. Parallel network simulations with NEURON. *Journal of Computational Neuroscience*, 21(2):119–129, 2006.

[36] D. Nicol. The Cost of Conservative Synchronization in Parallel Discrete Event Simulations. *J. ACM*, 40(2):304–333, Apr. 1993.

[37] D. Nicol and P. Heidelberger. Parallel Execution for Serial Simulators. *ACM Trans. Modeling and Comput. Simulation*, 6(3):210–242, July 1996.

[38] E. Painkras, L. A. Plana, et al. SpiNNaker : A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

[39] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha. Compass: A scalable simulator for an architecture for cognitive computing. In *ACM SC '12*, pages 1–11, Nov 2012.

[40] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *arXiv preprint arXiv: ...*, abs/1404.7:66, 2014.

[41] F. Schürmann, F. Delalondre, et al. Rebasing i/o for scientific computing: Leveraging storage class memory in an ibm bluegene/q supercomputer. In *ICS 15' Conference Proceedings*, ISC 2014, pages 331–347, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[42] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis. Cfd vision 2030 study: A path to revolutionary computational aerosciences. Technical Report NASA/CR-2014-21878, NASA, March 2014.

[43] R. Stein, A. S. French, and A. Holden. The frequency response, coherence, and information capacity of two neuronal models. *Biophysical journal*, 12(3):295–322, 1972.