Speculative Distributed Simulation of Very Large Spiking Neural Networks

Adriano Pimpini*
pimpini@diag.uniroma1.it
Sapienza, University of Rome
Rome, Italy

Andrea Piccione*
piccione@diag.uniroma1.it
Sapienza, University of Rome
Rome, Italy

Bruno Ciciani
ciciani@diag.uniroma1.it
Sapienza, University of Rome
Rome, Italy

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it
University of Rome “Tor Vergata”
Rome, Italy

ABSTRACT

Spiking Neural Networks are a class of Artificial Neural Networks that closely mimic biological neural networks. They are particularly interesting because of their potential to advance research in several fields, both because of better insights on neural behaviour (benefiting medicine, neuroscience, psychology) and the potential in Artificial Intelligence. Their ability to run on a low energy budget once implemented in hardware makes them even more appealing. However, because of their behaviour that evolves with time, when a hardware implementation is not available, their output cannot simply be computed with a one-shot function (however complex), but instead they need to be simulated.

Simulating Spiking Neural Networks is exceptionally costly, mainly due to their sheer size. Many current simulation methods have trouble scaling up on more powerful systems because of conservative synchronisation methods. Scalability is often offered through approximation of the actual results. In this paper, we present a modelling methodology and runtime-environment support adhering to the Time Warp synchronisation protocol, which enables speculative distributed simulation of Spiking Neural Network models with improved accuracy of the results. We discuss the methodological and technical aspects that will allow effective speculative simulation and present an experimental assessment on large virtualised environments, which shows the viability of simulating networks made of millions of neurons.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation**; **Distributed simulation**; • **Hardware** → **Neural systems**; • **Software and its engineering** → *Synchronization.*

KEYWORDS

Spiking Neural Networks, Parallel Discrete Event Simulation, Speculative Simulation, Time Warp.

*Both authors contributed equally to the paper.

ACM Reference Format:
Adriano Pimpini, Andrea Piccione, Bruno Ciciani, and Alessandro Pellegrini. 2022. Speculative Distributed Simulation of Very Large Spiking Neural Networks. In *SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '22), June 8–10, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3518997.3531027

1 INTRODUCTION

The interest in Spiking Neural Networks (SNNs) has increased in the last decade [19]. Their momentum is strongly related to their expressive capabilities, as they allow them to mimic biological neural networks closely. This characteristic makes SNNs a perfect tool for research activities in disparate fields, such as medicine, neuroscience, or psychology, and provides a non-negligible potential in Artificial Intelligence. At the same time, their significantly-reduced energy requirements open the way for specialised hardware applications in the form of *neuromorphic chips* [21, 38, 40]. These chips are regarded as one of the essential future steps in computing, as they introduce a level of parallelism, with reduced energy demand, that does not exist in today's hardware, including GPUs, FPGAs, and most AI accelerators.

Neuromorphic systems, in general, have an increased value due to their capability to perform processing asynchronously. Indeed, they rely on event-driven processing models to tackle complex computing problems, in a way similar to the human brain, which uses only a subset of its neurons and synapses to carry out tasks at maximum efficiency.

SNNs encode data in a temporal domain known as the *spike train* [6]. Due to this behaviour that evolves with time, their output cannot be simply computed with a one-shot function, however complex, but instead they need to be simulated. The event-driven processing model of SNNs makes them a perfect match with Discrete Event Simulation (DES) techniques. Nevertheless, simulating SNNs is exceptionally computationally intensive due to their sheer size and scale. This is reflected in non-negligible running times, making it difficult to obtain relevant simulation results reasonably in time. As an example [33], simulating 250ms of activity for a network of 11,250 neurons/127 million synapses on the well-known NEST simulator [18] can take more than 30 seconds on a single CPU core. At the same time, the research community is striving to simulate networks of size comparable to the mammalian brain's, containing on the order of 10^7 to 10^{11} neurons, with thousands of synapses per neuron on average [2, 24, 25, 28], which are considered very large networks.

The computational demand of large-scale SNN simulation is addressed by the vast majority of existing simulators by employing parallel/distributed execution. This is done by exploiting multicore CPUs (also in distributed environments) or by relying on accelerators such as GPUs [4, 7, 13, 15, 26, 32, 33, 52] or FPGAs [12, 47, 51]. Nevertheless, we observe that the trouble at scaling up faced by most state-of-the-art SNN simulation methods may be due to their use of conservative synchronisation. Indeed, in parallel/distributed simulation terminology, the typical execution scheme adopted by SNN simulators is to rely on Parallel Discrete Event Simulation (PDES) with a synchronous conservative synchronisation scheme using the YAWNS algorithm [34].

However, SNN models have, in general, a low rate of neuron activity at any given time, which is strictly connected to their brain-inspired nature, which uses only a subset of the neurons for a given task. This space/time partitioning of the activities makes it perfectly suitable for exploiting speculative simulation adhering to the Time Warp synchronisation protocol [27]. This choice could produce non-negligible simulation speedups, especially on large-scale computing infrastructures, as it has been shown that speculative PDES can be deployed on millions of (distributed) CPU cores [3].

At the same time, spikes carry a piece of information which is either binary (i.e., a neuron has spiked) or with a tiny payload (e.g., the intensity of the spike) depending on the nature of the model and its implementation—in general, the behaviour of spikes is somewhat homogeneous in a simulation. Similarly, the state of each neuron is reduced in size, and the time complexity of the execution of a single event can be significantly fine-grained. These aspects could make the employment of Time Warp-based PDES suboptimal, as the housekeeping cost might not be paid off by forward-processing activities [17].

In this paper, we present modelling methodologies and PDES runtime-environment support for SNN models based on the ROOT-Sim Simulation Framework [37], which employs the Time Warp synchronisation protocol to enhance the scalability of simulations. Our approach has a twofold goal. On the one hand, we tackle the complexity of deploying an SNN model on top of a speculative PDES runtime environment. The modeller should not need to worry about where the synapses are kept or how they are organised, nor should they care about the fact that spikes are, in fact, events that need to be sent (logically) one by one or, even worse, they should not need to schedule an event to check whether a spike should happen or not. Ideally, the modeller should be tasked with as few programming related actions as possible, made as simple as possible.

On the other hand, we focus on simulation accuracy. Indeed, many approaches relying on (time-stepped) synchronisation algorithms consider a fixed lookahead (typically set on the order of tenths of milliseconds). In this way, the simulation results *approximate* the actual results. This is a well-known limitation [23] related to classes of neuron models with linear subthreshold dynamics, wherein some circumstances even some spikes can be missed. Our modelling methodology leverages the nature of discrete events compared with an innovative numerical method and ad-hoc events management strategies, allowing us to obtain precise simulation results. Also, we apply this approach to exponential synapses, which are much more complex than jump synapses typically dealt with in the literature.

We released our implementation as open-source software[1]. The remainder of this paper is organised as follows. Section 2 discusses our research background, while Section 3 presents related work. Our simulation methodology and related runtime support are discussed in Section 4. The results of our experimental assessment are provided in Section 5.

2 BACKGROUND AND MOTIVATIONS

In this section, we discuss the background and motivations of our proposal. We first provide a recap of Time Warp synchronisation in Section 2.1, while in Section 2.2 we highlight the essential characteristics of SNNs, emphasising the hindrances when transitioning these models to Time Warp-based PDES runtime environments.

2.1 Time Warp Synchronisation

In PDES, the simulation model is partitioned into multiple Logical Processes (LPs), each associated with its own view of simulation time, known as *Local Virtual Time* (LVT). The LVT tracks the computation advancement locally along the logical-time axis. The execution of an event can mark an LP state update and makes the LVT jump to the timestamp of the processed event. It is the responsibility of the underlying runtime environment to track changes of the LVT of some LP when dispatching the event to be processed [16].

While processing an event, new timestamped events destined to any concurrent LP can be generated and injected into the system. At the same time, state transitions caused by event processing at the different LPs can occur concurrently.

Classical PDES relies on correctness rules that enforce non-decreasing timestamp-ordered state transitions at the different simulation objects [16]. Under this enforcement, each LP in the system has a coherent view of the flow of logical time.

In the speculative (optimistic) approach to synchronisation, also known as Time Warp [27], events are typically stored by the runtime environment into per-LP event lists, each of which is logically partitioned into a *future-event list* and a *past-event list*. The future-event list stores events not yet processed, while the past-event list records already-processed events. Each LP is eligible for dispatching along some thread running within the PDES platform unless its future-event list is empty. Once dispatched, the LP can process the minimum-timestamp event kept by its future-event list. Such an event is then moved to the past-event list.

In Time Warp, an LP is scheduled for execution without any safety verification of causal consistency of its next-to-be-processed event. Hence, timestamp-order violations might arise since an LP may receive an event (produced by another LP) with a timestamp lower than its LVT. If a timestamp-order violation is detected, all the events executed out of timestamp order are rolled back by the runtime environment—they are moved back from the past-event list to the future-event list. Also, the LVT of the LP is pushed back to the timestamp of the last event executed in the correct order, and the LP's state is restored to its value prior to the timestamp order violation, which is achieved by either relying on traditional

[1]The source code and reproducibility package is available at https://doi.org/10.5281/zenodo.6475218.

checkpointing methods (see, e.g., [42, 43]) or through reverse computing approaches (see, e.g., [9, 14]), where reverse versions of the event processing routines are executed when rolling back processed events.

Restoring the system to a correct state entails restoring individual LP states independently, with no risk of a domino effect. In particular, if dependencies occurred due to the scheduling of some event between a rolling back LP_a and another one LP_b, these dependencies are undone via so-called *anti-events*[2]. More in detail, an anti-event is generated for each event injected by LP_a during the portion of the computation to be rolled back, and is used to retract the initially injected event. The recipient LP_b also rolls back after receiving an anti-event associated with an already-executed event. Instead, if the event has not yet been executed, the anti-event has the only effect to "annihilate" the initially injected event. This operation typically occurs within the runtime environment. After rolling back, any LP resumes the execution of the events from its future-event list.

2.2 Spiking Neural Networks

Spiking Neural Networks (SNNs) are a class of Artificial Neural Networks (ANNs) that closely mimic *natural* neural networks. This capability is achieved by using spiking neurons, which communicate by sending signals (spikes) through synapses. Spiking neurons are *stateful*, and the synapses connecting them can be too.

Unlike what happens in other classes of ANNs, spiking neurons fire only when their *membrane potential* reaches a particular threshold value. When a spiking neuron fires, it generates a spike propagated to the neurons it is connected to, which react by increasing or decreasing their membrane potential accordingly, over time. However, before reaching other neurons, the spike passes through synapses, which are weighted and introduce a *transmission delay*. Indeed, a fundamental aspect that differentiates SNNs from other ANNs is the role time plays. This leads to a higher fidelity execution of the neural network at the price of higher computational costs.

Spiking-neuron models are derived from experimental observation of natural neurons' behaviour. Since the neurons react to and communicate through electrical stimuli, they can be modelled as circuits. The structure and parameters of the circuits are derived by feeding the neuron with different input currents and seeing the response to the various stimuli. We know that neurons' plasma membrane isolating properties give rise to a capacitance (membrane capacitance C_m) and that the potential between the two sides of the membrane (that we refer to as membrane potential V_m) is what kick-starts the action potential propagation once it reaches a target threshold value V_{th}. Furthermore, we know that in the absence of stimuli, the membrane potential resets to a resting value V_r; this also holds after the action potential is generated (which we also refer to as firing or spiking) and the sodium-potassium pump is done reverting the neuron to its resting state, that is, after the refractory period τ_{ref} is elapsed.

Additionally, for the membrane potential to rise, there has to be some kind of input current I, which is the sum of the stimuli coming from presynaptic neurons, and that an external current I_{ext} can be supplied (e.g. for experimental observation). This series of observations gives some clues about what to look for when creating a biological neuron model. Furthermore, the capacitance alone makes it evident that a spiking neuron is stateful (the minimum state being just the membrane potential at a given time), and such a state evolves with time. This hints at a fundamental aspect of SNNs: to be run on computers, networks of spiking neurons have to be simulated through time. This means that running an SNN on a computer can be a costly endeavour.

If running an SNN is so much more computationally expensive than other ANNs, why should we focus on them? The first reason is obvious and possibly already satisfactory on its own: we want to eventually be able to efficiently and precisely simulate a human brain (or parts of it) to be able to study in detail its behaviour in various experiments or to understand how modifications in the structure or physical aspects of its components would impact it. Neurological research would gain a powerful tool to test and validate hypotheses; medicine would be helped in diagnosing and treating brain diseases.

The second reason is that, since spiking neurons are modelled as electronic circuits, they can easily be implemented in hardware. A series of *neuromorphic chips* have already been commercialised [1, 10, 11]. While the design of these chips can directly benefit from simulation studies, their hardware implementation becomes incredibly cost-effective, also energy- and performance-wise. Additionally, a series of advantages arise with respect to all other ANNs. First and foremost, neuromorphic chips suffer from no approximation: since the electronic components are physically present, there is no approximation stemming from the precision limit inherent in computer simulations. Second, the computation is inherently and naturally parallel, making the execution of parallel algorithms more effective (e.g., compared to the synchronisation cost paid on more traditional Von-Neumann computing architectures).

Finally, current runtime environments for SNNs simulations are mostly time-stepped, carrying out the simulation by computing the state of all neurons/synapses at every small increment of time. Updating all states at each timestamp means updating objects that are doing nothing, too, introducing costs that could be avoided. Moreover, the time step (which is typically fixed in a simulation run) determines the actual accuracy of the simulation results. Conversely, by resorting to speculative PDES, the accuracy of the simulation is increased thanks to the timestamp advancement, which better captures the evolution of simulation time. Moreover, it is possible to use the available computing resources better, ignoring inactive neurons at any given simulation phase.

3 RELATED WORK

Several works have proposed techniques to speed up or scale out SNN simulations. Notably, in [39], the authors have shown that relying on speculative PDES simulation using the Time Warp synchronisation protocol can lead to non-minimal performance improvement, especially in scenarios where only a subset of neurons, in a specific time window, are actively sending spikes to each other. While this work mainly focuses on the TrueNorth Leaky Integrate and Fire (TNLIF) [11] architecture, the seminal results in this paper can be deemed general. We complement the results in this

[2]An anti-event is a "negative copy" of the corresponding event, or of its digest.

work by showing techniques that allow providing performance improvements when a large number of neurons are active, and by experimentally studying the performance when varying several parameters of the SNN models.

In [33] the authors show the viability of running SNN simulations on top of heterogeneous hardware, possibly composed of CPUs, GPUs and FPGAs. The main contribution in this work is identifying portions of code in an SNN model that is repeatedly used in multiple simulations and/or runtime environments. This code is manually ported to the different hardware architectures. Conversely, the code specifically bound to the model is automatically transformed towards the target architecture. While we do not explicitly target heterogeneous hardware in our proposal, we leverage the idea that there is a clear separation between the runtime support and the actual SNN model. This allows us to introduce several optimisations at the runtime environment level to benefit multiple simulation scenarios.

Several works (see, e.g., [23, 30, 35]) have tackled the accuracy of SNN simulations by either numerical approaches or parameter estimation. In our contribution, we follow the path of bridging the gap between continuous simulation and discrete-event simulation by showing that, by manipulating differential equations to compute the next-spiking time, it is possible to exploit several optimisations at the runtime environment level to deliver non-negligible scalability improvements. This is a research path already explored in [49], although we show that this technique can also lead to interesting scalability results in worst-case scenarios and using more complex instantaneous raise/exponential decay synapses, with proper management of the discrete events by the runtime environment.

Some of the techniques introduced in our PDES environment have already been proven valuable in other scenarios. In particular, we extensively rely on the capability of the runtime environment to *retract* already-injected events without invoking any code from the SNN model. In [20], the authors have shown that this approach can deliver significant performance improvements also when simulating large biochemical networks. We also rely on the capability of the PDES runtime environment to deliver the same event to multiple destination LPs in a *publish/subscribe* fashion. This approach is clearly related to one-to-many communication primitives offered by the MPI specification [22], although we have to consistently take into account the possibility that some or all the events injected in the simulation according to this scheme might be subject to rollbacks. Carefully managing these rollbacks has a direct effect on the overall performance.

The relevance of large-scale SNNs simulation is also reflected in the availability of multiple tools to design experiments and obtain simulation data. Several SNN simulators have been proposed in the literature, frequently focusing on scalability to large networks.

Among them, Brian [48], Neuron [8], and NEST [18] are the most promising ones concerning either simulation performance or usability—they all offer some form of Python bindings. At the same time, a common feature for these simulation runtimes environments is that they use time-stepped simulation algorithms with limited lookahead.

Brian [48], implemented in python, has ease of use as its primary focus while sporting a series of interesting facilities that allow for high flexibility and performance. The configuration of a model run on Brian is based on a string representation of the differential equations describing the neuron/synapses state and their evolution. These strings are parsed using SymPy [31], a library for symbolic mathematics. The code to run the simulation is compiled on the fly when initialising the simulation. Nevertheless, the execution is purely sequential at the present date.

Neuron [8], written for the most part in C and C++, has its main focus on biologically-accurate simulation. In this sense, it provides facilities to observe also internal aspects of neurons and synapses, such as the model's electrical, chemical, and topological evolution. The simulator can describe ion concentrations and the functioning of ion gates, the dynamics of ion diffusion, and more. It ships with a 3D library to model various parts of the cell soma, dendrites, axons, along with their 3D position and orientation, both in the neuron and in space.

On the deployment side, Neuron supports running simulations on clusters through MPI. However, when running on multicores, Neuron does not exploit the worker thread paradigm, but the various kernel instances are run as separate processes, and MPI is used to connect them, leading to a worse performance with respect to what would be achieved using worker threads and using MPI only to reach physically remote nodes.

NEST [18] comes prepacked with "over 50 neuron models, many of which have been published" and "over 10 synapse models" that can also be used to implement new custom neuron and synapse models. NEST can run parallel simulations through OpenMP. Distributed simulations are also supported, and MPI is used to take care of message passing between multiple computational nodes. Neurons are instantiated only on the node on which they belong, while synapses are handled at the receiving node's end for matters of synapse plasticity.

Other established simulators, such as CARLsim [13], NCS [26], NeMo [15], Nengo [4], HRLSim [32], and PCSIM [36] support multi-threaded execution on CPUs, some of them with support for execution on multiple nodes. GeNN [52] can be executed on a single GPU. Some simulators additionally support the execution in GPU clusters [13, 15, 26, 32, 52]. CARLsim [13] and NCS [26] support a CPU-GPU co-execution.

4 SIMULATING LARGE SNNS

An SNN simulation can be seen as a collection of simulations of individual neurons that interact by the exchange of spikes. The changes in neuron state may trigger the emission of a spike delivered to the connected neurons. Since the spikes must be considered in the target neurons' future state updates, a neuron's state can only be consistently updated once it has received all spikes with smaller timestamps.

The computation of neuron state updates and the delivery of the generated spikes are the two principal operations implemented by SNN simulation platforms. Since most of them employ a timestep-driven approach, commonly used neuron models have been picked favoring those handily computable via some iterative numerical procedure, such as the Euler method.

Anyhow, the conservative synchronisation scheme typically employed by SNN simulators cannot efficiently deal with arbitrarily-low synaptic delays, which would otherwise force a costly synchronisation and spike delivery operation after each timestep. For this reason, many models include a minimum fixed delay in spikes transmission to reduce the necessary synchronisation steps and improve spike delivery performance.

Many SNN models having been formulated with mainly the time-stepped approach in mind. Moreover, to the best of our knowledge, the only SNN synapse model used in (P)DES are jump synapses. When a jump synapse transmits a spike, it is instantaneously applied to the membrane in a Dirac-delta fashion, resulting in a jump in the membrane potential. This behaviour is highly convenient for PDES simulations as it requires no further computation and perfectly fits the DES paradigm: if a spike results in the membrane potential surpassing the threshold, the neuron spikes, otherwise it does not.

In this work, we transition to the optimistic PDES paradigm *instantaneous raise/exponential decay synapses*, commonly called just exponential synapses. This type of synapse does not directly act on the membrane potential, but rather, it generates an instantaneous increase in the incoming current the neuron receives. The current's effects are applied over time to the membrane potential: the latter rises over time, charging similarly to an electronic capacitor. At the same time, the current's intensity decreases exponentially with time. This means the neuron might spike in the future, the hypothetical spike time (if any) has to be computed via numerical methods, and the resulting event enqueued.

In the following, we explain the steps needed to represent this spiking model in a PDES runtime successfully, and we then describe the extensions to the traditional PDES facilities required to speed up the spike delivery for SNNs.

4.1 Neuron Models for PDES SNN Simulations

Neuron models are commonly expressed as a set of differential equations that describe the behaviour of its membrane potential and synaptic currents. Given a known neuron state, generating discrete-event spikes require computing the next spike timing. Therefore we need to solve the state equations explicitly.

While the modeling approach that we present here is general, we exemplify it by showing how we can manipulate a common kind of neurons to be simulated on top of a speculative PDES runtime environment adhering to the Time Warp synchronization protocol.

We focus here on the Leaky Integrate and Fire (LIF) neuron model, which, conveniently, is also one of the most commonly used in large SNN simulations. Equations (1) describe the subthreshold dynamics of the neuron, that is the evolution of its state in the absence of emitted spikes, where $V(t)$ is the membrane potential, and $I(t)$ is the current flowing inside the neuron across the membrane.

$$\begin{gathered}\frac{\mathrm{d}V(t)}{\mathrm{d}t} = \frac{-V(t) + V_r}{\tau_m} + \frac{I(t) + I_{ext}}{C_m} \\ \frac{\mathrm{d}I(t)}{\mathrm{d}t} = -\frac{I(t)}{\tau_{syn}}\end{gathered} \tag{1}$$

The positive quantities τ_m, τ_{syn}, C_m represent the membrane time constant, the synaptic time constant and the membrane capacitance, respectively. The quantity I_{ext} is a constant external current stimulus, while V_r is the reset potential. For a more thorough discussion on the meaning of these parameters, the reader can refer to [6].

The non-linear spike behaviour works as follows: if, at any time t, $V(t)$ overcomes a voltage threshold V_{th}, the neuron emits a spike, then $V(t)$ is forcefully reset at voltage V_r for a period of time τ_r, the so-called refractory period. Spikes are delivered to post-synaptic neurons with a delay in virtual time and an effect established by the synapse model. Many large simulations employ a simple synapse model, characterised by a fixed transmission delay t_{trans} and a weight w. By using this simple model, a spike causes the post-synaptic neuron to instantaneously increase its $I(t)$ by w.

Solving Equations (1) in $V(t)$ and $I(t)$ yields Equations (2).

$$\begin{gathered}V(t) = I_0 A_1 e^{-\frac{\Delta t}{\tau_{syn}}} + (V_0 - A_2 - I_0 A_1) e^{-\frac{\Delta t}{\tau_m}} + A_2 \\ I(t) = e^{-\frac{\Delta t}{\tau_{syn}}} I_0\end{gathered} \tag{2}$$

In these equations, V_0 and I_0 represent the state of the neuron at time t_0, while the two constants A_1 and A_2 have been introduced for the sole purpose of readability: their expansions are found in Equations (3). Constants A_0 and A_2 can be computed once at simulation startup.

$$\begin{gathered}A_1 = \frac{1}{\left(\frac{1}{\tau_m} - \frac{1}{\tau_{syn}}\right) C_m} \\ A_2 = V_r + \tau_m \frac{I_{ext}}{C_m}\end{gathered} \tag{3}$$

To compute the next spike timing, it would be sufficient to solve Equations (2) in t with $V(t) = V_{th}$. Unfortunately, the general case solution is not analytical, we must therefore resort to relatively expensive numerical methods. We want to avoid their use; therefore, we will now carry out further analysis.

It is possible to distinguish between self-spiking and nonself-spiking neurons. If, for a given neuron, its set of parameters satisfies the condition in Equation (4), the neuron is self-spiking. In other words, it spikes periodically on its own, without needing to receive spikes in input.

$$\lim_{t \to \infty} V(t) = A_2 > V_{th} \tag{4}$$

In this case, using the bisection method, we can compute once at startup the constant τ_{self}, the self spike timing in absence of synaptic inputs, i.e. $I(t_0) = 0, V(t_0) = V_r$. To find a suitable window for the bisection method, we can simply consider the interval $[t_0, t_{large}]$, with t_{large} chosen large enough so that $V(t_{large}) > V_{th}$: such a value must exist, since the condition in Equation (4) holds and, with these conditions, $V(t)$ is monotonic increasing.

For a nonself-spiking neuron, assuming the absence of incoming spikes, we can derive a simple procedure to establish whether it will emit a spike in the future. In Equations 5 we provide the definition and the explicit value of the constant I_{th}. The monotonic decreasing property of the $I(t)$ state equation guarantees that $I_0 > I_{th}$ is a necessary (but not sufficient) condition for a nonself-spiking neuron to spike in the future.

$$\begin{gathered}I_{th} := I(t) \mid V(t) = V_{th} \wedge \frac{\mathrm{d}V(t)}{\mathrm{d}t} = 0 \\ I_{th} = C_m \frac{V_{th} - V_r}{\tau_m} - I_{ext}\end{gathered} \tag{5}$$

If a nonself-spiking neuron satisfies the necessary spike condition, we can compute t_{th}, the time needed to decay from I_0 to I_{th}, as shown in Equation 6. If $V(t_{th}) < V_{th}$ then we can conclude that the neuron does not spike, otherwise it will spike at time $t_{spike} \in [t_0, t_{th}]$.

$$t_{th} = -\ln\left(\frac{I_{th}}{I_0}\right)\tau_{syn} \tag{6}$$

This result holds because we have $V(t_0) < V_{th}$ and $V(t_{th}) \geq V_{th}$ therefore, given the continuity of the $V(t)$, at least one $t_{spike} \in [t_0, t_{th}]$ must exist. We thus compute t_{spike} using the bisection method—we noted that higher-order numerical methods such as the Newton method are not always numerically stable during our experimental phase. Otherwise, if $V(t_{th}) < V_{th}$, then $V(t) < V_{th}$ for all $t \in [t_0, t_{th}]$: the neuron did not and will not spike in absence of further stimuli since $I(t) < I_{th}$ for all $t > t_{th}$.

The proof that t_{spike} is unique in the $[t_0, t_{th}]$ interval is more involved, so we will only sketch the main idea. Using the definition of I_{th} it is possible to show that $\frac{\mathrm{d}V(t)}{\mathrm{d}t} = 0$ for exactly one $t_d \in [t_0, t_{th}]$, where $V(t)$ reaches its maximum. We have that $\frac{\mathrm{d}V(t_0)}{\mathrm{d}t} > 0$ and $\frac{\mathrm{d}V(t_{th})}{\mathrm{d}t} \leq 0$. Also, since $\frac{\mathrm{d}V(t)}{\mathrm{d}t}$ is continuous, we can conclude that $V(t)$ is monotonous increasing in $[t_0, t_d]$ and monotonous decreasing in $[t_d, t_{th}]$; t_{spike} is therefore unique and lies in the interval $[t_0, t_d]$.

For self-spiking neurons, we assume that most of the time, they will spike approximately every τ_{self} seconds: so we search the spike window by increasing tentatively t_{th} by τ_{self}. To compute a precise t_{spike}, we use the bisection method as well.

4.2 From Neurons to Logical Processes

To transition an SNN simulation to the optimistic PDES paradigm, we must partition its state into independent Logical Processes (LPs). In this work, we assume that synapses state is static, which is not a very limiting factor since many large SNN models in the literature embrace this limitation. With this in mind, the natural way to map neurons to LPs would be a one-to-one mapping. Clearly, it is possible to collect together more neurons inside a single LP, but we expect that such a strategy may only be worthwhile if it can be guaranteed that neurons grouped inside the same LP are tightly connected; or else, for example, a straggler event directed at a neuron would cause the rollback of other neurons grouped with it that would not have otherwise rollbacked. Therefore in this work, also for simplicity reasons, we employed the natural one-to-one mapping. Nonetheless, we think that the potential trade-off offered by a non-trivial neuron-to-LP mapping is an interesting one that may be featured in future work.

In the following sections, we will assume that we target a distributed simulation environment running on multiple computing nodes, each hosting several processing threads. We also assume, for the sake of simplicity, that the binding of LPs to processing threads is established at simulation startup and fixed for the simulation duration.

4.3 Dynamic Spiking Events

We have shown earlier how to determine the next spike timing t_{spike} for a LIF neuron with the condition that, in the meanwhile, such neuron does not receive any spike. In a discrete event simulation, we would schedule a new event E with time t_{spike} which represents the spike potentially emitted in the future. If the neuron receives a spike before t_{spike}, its previously computed spike timing would not be consistent anymore with the newly induced state change, and therefore we would need to somehow invalidate or retract the event E.

A working solution to this problem consists in augmenting the neuron state with an *epoch* number, which is increased every time the neuron receives a spike. The potential spike emission events are tagged with the current epoch number. When a potential spike emission event has to be processed, the model checks if the neuron epoch number is consistent with the event's one. If it is not, it is simply ignored and discarded. While, on paper, this approach seems reasonable, its application to SNN simulations fails miserably. Each neuron state change would schedule a new future spike event, and, given the high out-degree of neurons (as mentioned earlier, in the order of thousands), every time a neuron were to actually spike, all of the receiving neurons would have to update their states: the event queue would end up being hogged by invalidated spike events. Most of the processing time would be spent discarding invalid events at the model level.

Our proposed solution consists in providing the model developer with *dynamic spiking events*, i.e. with the possibility of scheduling *retractable events* i.e. events that can be arbitrarily removed from the events queue or whose timestamp can be changed at will. These operations are supported at the runtime environment level. Future spike emission events can be effectively represented as retractable events. In the context of SNN simulations, each neuron needs a maximum of one retractable event destined to itself. This consideration significantly simplifies the implementation of this new mechanism. Each thread simply has a private queue that handles the retractable events associated with the neurons bound to it. The private queue is implemented as a k-heap data structure which allows efficient priority changes and removal of events. In order to extract a new event, each thread chooses the lowest timestamp between the normal events queue and its private retractable events queue.

In order to correctly handle rollbacks, when computing the checkpoint of an LP, we must also include their currently active retractable event. With this precaution, when an LP must roll back to a previous state, it can correctly restore its retractable message.

4.4 Publish/Subscribe Events

The use of retractable messages effectively reduces the number of events delivered at the simulation model during the simulation run, but it does not impact the number of events used to convey the actual spikes. Each time a neuron spikes, many events corresponding to the out-degree of the neuron must be generated and delivered one by one to the proper LP, possibly over the network if the targeted LP is residing on another physical node. An early experimental assessment showed that this was a significant performance bottleneck.

For this reason, we came up with a new feature that allows an LP to *publish* special events to which other LPs may *subscribe*. During simulation initialisation, neurons can subscribe to the ones they are

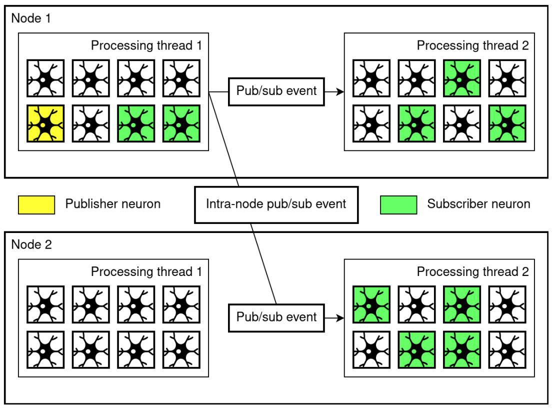


Figure 1: Publish/subscribe events in a distributed environment. In this example, the publisher LP generates only two events, instead of nine.

connected to. During the actual simulation, neurons will emit spikes in the form of published events. With this contract in place, the simulation runtime can significantly reduce the number of events actively transmitted around.

To simplify our system implementation's description, we say that a processing thread is subscribed to an LP if any bound LPs are subscribed to it. Similarly, a computing node is subscribed to an LP if any hosted processing thread is subscribed to it. Our implementation expects a single function that specifies the publish/subscribe graph. Each processing thread involved in the simulation initialises the internal data structures based on the relevant subset of information in the graph. In particular, each LP maintains a list of references to the local processing threads and remote nodes subscribed to it. Also, each node hosts a global table that maps identifiers of publisher LPs to a list of references to the local processing threads and related LPs subscribed to it.

As exemplified in Figure 1, publish/subscribe events can reduce the number of events exchanged by processing units dramatically. When an LP publishes a new event, the simulation framework only generates and delivers the events as specified in its subscription list, sending one copy per local subscriber thread and one per remote subscribed node, minimising the number of messages transferred, both locally and over the network. When a processing thread extracts a publish/subscribe event, it simply needs to get the list of subscribed LPs from the global table to forward them a copy of the event.

The management of the rollback operation for publish/subscribe events can be realised according to the traditional scheme supported by Time Warp synchronisation, i.e. by relying on anti-events. Indeed, publish/subscribe anti-events are no different from regular events, especially after a single publish/subscribe event is materialised into multiple copies for each subscribed LP. Our implementation keeps track of the local events generated by publish/subscribe events so that their anti-events do not need copies delivered to each subscribed LP. This helps reduce the overhead of anti-events delivery and also simplifies fossil collection operations.

Table 1: Connectivity map for the synthetic benchmark.

		to					
		L1e	L1i	L2e	L2i	L3e	L3i
from	In	0.292	0.192	0.049	0.237	0.169	0.115
	L1e	0.224	0.293	0.106	0.254	0.438	0.099
	L1i	0.135	0.025	0.409	0.25	0.309	0.271
	L2e	0.165	0.177	0.122	0.032	0.491	0.3
	L2i	0.448	0.319	0.08	0.207	0.225	0.201
	L3e	0.395	0.123	0.265	0.215	0.476	0.174
	L3i	0.223	0.276	0.358	0.028	0.065	0.188

5 EXPERIMENTAL RESULTS

We present an experimental assessment of the proposed approach. The results have been obtained by relying on a set of virtual machines on Amazon Web Services. In particular, we have used various configurations of m5.4xlarge instances (equipped with 16 virtual cores), m5.8xlarge instances (equipped with 32 virtual cores), and m5.24xlarge instances (equipped with 96 virtual cores). Our proposal has been implemented within the ROme OpTimistic Simulator (ROOT-Sim) [37]. We compare our results with both Brian and NEST, both from a correctness point of view and from a performance/scalability perspective. All results are averaged over 5 different runs. The same sequence of random numbers has been used in the experiment.

5.1 Benchmarks

To study our proposal's correctness and performance/scalability, we have considered two synthetic SNN models, one real-world model, and a standard benchmark used in the literature.

The first synthetic benchmark is a network of 1,000 LIF neurons, divided into one input layer and three "passive" layers, each of which has two populations, one excitatory and one inhibitory. The input layer has 100 excitatory LIF neurons, each of which receives a constant current input, and each of the three layers has 200 excitatory and 100 inhibitory LIF neurons. It employs a connectivity map (reported in Table 1) to connect the populations. This configuration aims to model a small network to compare the data for correctness against state-of-the-art simulators.

The second synthetic benchmark is a network composed of only two neurons, an input one N_1 and an output one N_2. N_1 receives a constant external current of 1,800 mV, and its output is propagated to N_2 through an exponential synapse with a weight of 5,000. N_2 receives no other input. The output of N_2 is monitored, and its spikes are collected. The simplicity of this network allows us to evaluate simulation accuracy.

The real-world model that we consider is Potjans and Diesmann's [41], depicted in Figure 2. It is an accurate model of the local cortical microcircuit entailing four layers of the cortex, named 2/3, 4, 5, and 6, each with an excitatory and an inhibitory neuron population. Layers are connected according to the "*connection probability* of a connection" that "defines the probability that a neuron in the presynaptic population forms at least one synapse with a neuron in the post-synaptic population. A *connectivity map* is defined by the 64 connection probabilities between the 8 considered cell types". The synapses in the model are static. The number of neurons per population is chosen according to [5], totalling 77,169 neurons. Furthermore, every layer can receive a background input in the form

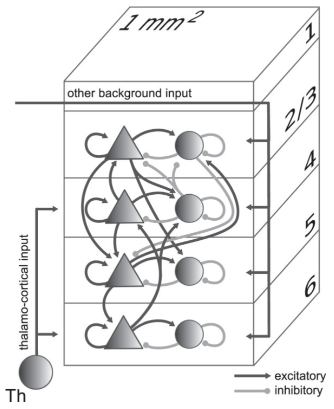
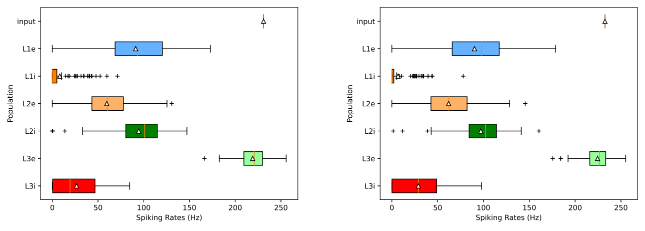


Figure 2: Potjans and Diesmann's Model definition. Excitatory populations are represented by triangles, and inhibitory populations are circles. Image taken from [41].



(a) ROOT-Sim. **(b) Brian.**

Figure 3: Spiking rates (Hz) for 1000 neurons model.

of a continuous current and input from an external thalamocortical neuron population. The thalamic neurons are implemented as Poisson neurons [46] with a fixed spiking rate.

The standard benchmark that we have used is inspired by a study on signal propagation in LIF models [50]. This benchmark [6] considers current-based (CUBA) synaptic interactions in a network of 300,000 LIF neurons, separated into two populations of excitatory and inhibitory neurons, forming 80% and 20% of the neurons, respectively. All neurons are connected randomly using a connection probability of 2%.

5.2 Correctness and Accuracy Results

To verify the correctness of our proposal, we have run the synthetic model both on ROOT-Sim and Brian. This benchmark allows us to validate the ensemble behaviour of the network and the behaviour on differently timed spikes. In Figure 3 we report a boxplot of spiking frequencies of every population. As we can see from the figure, the spiking frequencies closely resemble one another. The difference is likely related to the fact that we do not employ a time-stepped simulation, i.e. our results do not suffer from any approximation in the spiking time. This result shows how the LIF model presented in this paper can be deemed correct, even when employed in non-minimal networks with neurons handling spikes coming in varied patterns from many other neurons.

Table 2: Neurons and populations parameter specification.

Populations and inputs

Name	L2/3e	L2/3i	L4e	L4i	L5e	L5i	L6e	L6i	Th
Population size, N	20683	5834	21915	5479	4850	1065	14395	2948	902
External inputs, k_{ext}	1600	1500	2100	1900	2000	1900	2900	2100	n/a

Neuron Model

Name	Value	Description
τ_m	10 ms	Membrane time constant
τ_{ref}	2 ms	Absolute refractory period
τ_{syn}	0.5 ms	Postsynaptic current time constant
C_m	250 pF	Membrane capacity
V_{reset}	−65 mV	Reset potential
V_{th}	−50 mV	Fixed firing threshold
θ	15 Hz	Thalamic firing rate during input period

Table 3: Connectivity and Synaptic parameter specification.

Connectivity

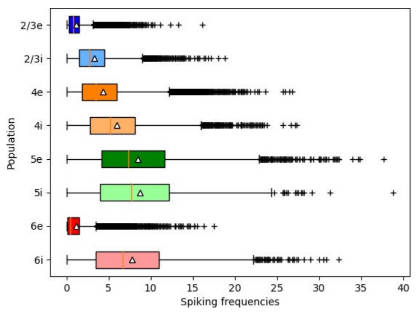
					from					
		L2/3e	L2/3i	L4e	L4i	L5e	L5i	L6e	L6i	Th
to	L2/3e	0.101	0.169	0.044	0.082	0.032	0.0	0.008	0.0	0.0
	L2/3i	0.135	0.137	0.032	0.052	0.075	0.0	0.004	0.0	0.0
	L4e	0.008	0.006	0.050	0.135	0.007	0.0003	0.045	0.0	0.0983
	L4i	0.069	0.003	0.079	0.160	0.003	0.0	0.106	0.0	0.0619
	L5e	0.100	0.062	0.051	0.006	0.083	0.373	0.020	0.0	0.0
	L5i	0.055	0.027	0.026	0.002	0.060	0.316	0.009	0.0	0.0
	L6e	0.016	0.007	0.021	0.017	0.057	0.020	0.040	0.225	0.0512
	L6i	0.036	0.001	0.003	0.001	0.028	0.008	0.066	0.144	0.0196

Name	Value	Description
$w \pm \delta w$	87.8 ± 8.8 pA	Excitatory synaptic strengths
g	−4	Relative inhibitory synaptic strength
$d_e \pm \delta d_e$	1.5 ± 0.75 ms	Excitatory synaptic transmission delays
$d_i \pm \delta d_i$	0.8 ± 0.4 ms	Inhibitory synaptic transmission delays

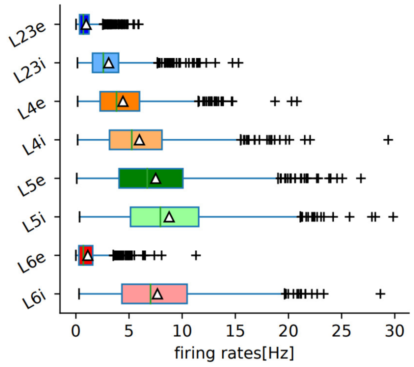
We have also considered the Potjans and Diesmann model to confirm these results. This model is highly relevant, with its implementation being part of the examples in the NEST simulator, and has been replicated in 2018 by Shimoura et al. [45] in an implementation for the Brian simulator. We have compared our implementation of this model with the implementations in both NEST and Brian. For the sake of comparison clarity, we report in Table 2 the neuron parameters (population size for each population, external inputs for e each population, and the neuron's physical properties), while Table 3 contains the connectivity map and the synaptic connection parameters used in all three models.

The simulation results are shown in Figure 4, again in the form of boxplots showing the observed spiking rates. Similarly to the previous benchmark, we observe that the results are extremely close. Again, the slight difference can be related to the increased accuracy that our solution can offer with respect to a time-stepped simulation.

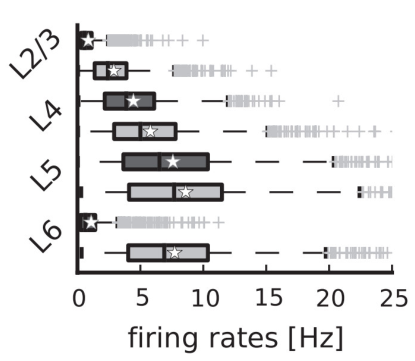
In Figure 5 we report the accuracy results for the 2-neuron model. In particular, for this simple model, we have generated the timing of N_2's spike train via numerical methods with a tolerance of 10^{-7} milliseconds . It has been used as a reference for the model simulated both on NEST and ROOT-Sim, where we have collected data for 1 second of simulated time. In the Figure, we report the absolute error for each i-th spike in the train. While the accumulated error grows linearly for both the simulation method used by NEST and ROOT-Sim, it is clear that the error in the NEST model is significantly higher, totalling 15 ms of error over 1 second of simulated neuronal



(a) ROOT-Sim.



(b) Brian (taken from [45]).



(c) NEST (taken from [41]).

Figure 4: Spiking rates (Hz) for Potjans and Diesmann's model.

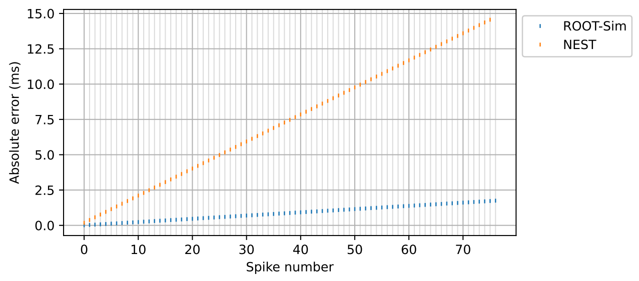
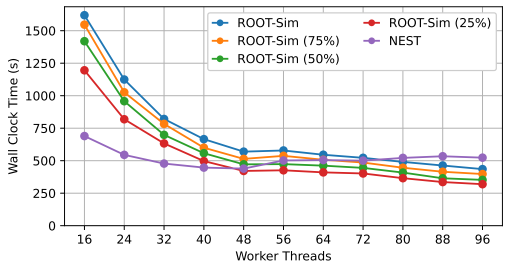
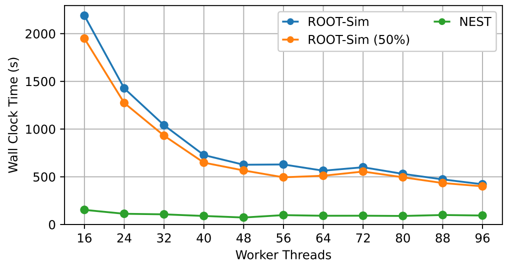


Figure 5: Accuracy results with a 2-neuron model.



(a) CUBA.



(b) Potjans and Diesmann's.

Figure 6: Single-node Scalability (m5.24xlarge).

activity (ROOT-Sim shows an error reduction of ~ 90% compared to NEST), with a simple network composed of only two neurons in a very short time window. This error is also reflected in the number of spikes—NEST cannot simulate the total number of spikes in the considered simulation window.

5.3 Performance Results

To assess the performance and scalability of our proposal, we have relied on the Potjans and Diesmann's and CUBA models, and compared against state-of-the-art NEST implementations. In Figure 6 we report the total execution time (in seconds) for both benchmarks when varying the number of threads on a 96-vCPU m5.24xlarge instance. We have also studied both simulators' performance when varying the total amount of interconnections in the network, i.e. running from the full load (as in the original benchmark) down to 25% of the total number of connections among neurons.

In both scenarios, we see that NEST performance is mostly independent of the number of threads used and the number of considered connections[3]. This is related to the approximated time-stepped nature of the simulation algorithm. Indeed, most of the time is spent by the NEST kernel advancing through the different steps and synchronising the various threads (through OpenMP [44]).

[3]For the sake of readability, we report only the curve related to the full connection configuration for NEST, but the results with a reduced density of the graph do not change at all.

Conversely, our implementation exhibits a performance improvement that grows with the number of parallel threads used. At maximum parallelism, our simulations deliver a performance improvement of 4x in both benchmarks. More interestingly, the scalability trend shows that the minimum in the simulation time curve has not been reached yet, suggesting that the amount of parallelism that the Time Warp simulation can exploit is still non-minimal.

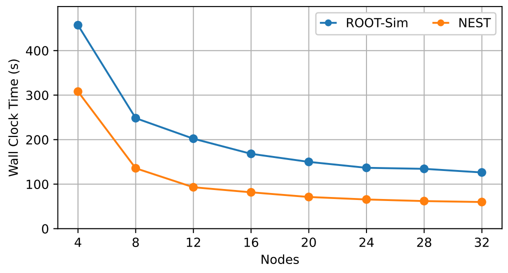
While the constant execution time of NEST consistently outperforms our implementation in the case of a smaller network (Potjans and Diesmann's model in Figure 6b), in the case of a larger network, NEST's performance starts to degrade at around 56 threads, when our solution starts to outperform it. This is an indication that, if larger networks were to be simulated, our solution could provide reduced simulation times—again, also offering results that are precise. This is a significant result, as the research community has the ambitious goal of "simulating the brains of mammals with a high level of biological accuracy and, ultimately, to study the steps involved in the emergence of biological intelligence" [29].

We have also studied the performance and scalability of our proposal, again against NEST, in a distributed environment, relying on MPI for both simulators. We have performed a strong scalability assessment, keeping fixed the size of the networks simulated by both benchmarks and varying the number of nodes from 4 to 32. We have used a set of m5.4xlarge instances (Figure 7) and a set of m5.8xlarge instances (Figure 8). These virtual machines are equipped with 16 and 32 virtual cores, respectively—we have used up to 512 distributed virtual cores, mimicking a tiny-scale supercomputer. By the results in Figure 6, this is a worst-case scenario for our implementation, as NEST is still outperforming ROOT-Sim on a single node in both configurations for both benchmarks.

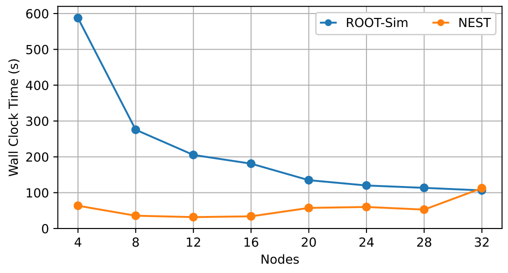
From the results, we anyhow observe that the scalability trends of both ROOT-Sim and NEST are comparable for the CUBA benchmark (Figures 7a and 8a). This is an indication that our proposal (particularly concerning retractable messages and publish/subscribe events) can dampen out the performance penalty observed on the single node in the case of a distributed simulation. Conversely, in the case of a smaller network (Figures 7b and 8b) the NEST implementation does not scale, probably due to synchronization overhead, although, in most of the configurations, it is still able to deliver better performance results—again, sacrificing preciseness.

To better understand the dynamics of our PDES speculative simulation with respect to NEST's time-stepped simulation, we have also carried out an experiment using the large network in CUBA benchmark when varying the network activity, i.e. the total number of spikes in the simulation. From the results reported in Figure 9, we see that NEST is also showing a performance that is independent of the amount of activity in the network (Figure 9b), while the Time Warp simulation based on ROOT-Sim can exploit this reduced amount of interactions. This is an expected result, given the nature of both simulation methodologies. At the same time, it is interesting to note that when the number of nodes is reduced, the benefit is increased (see Figure 9a). This is related to the fact that more LPs are bound to the same thread when the number of nodes is smaller. In this scenario, the publish/subscribe messaging mechanism that we have devised is likely to pay off more. This indicates that if larger networks have to be simulated on a same-scale machine, the performance improvement offered by this runtime support can be non-minimal.

Finally, to shed light on the reason behind the performance gap between the ROOT-Sim implementation and the NEST one (although we recall that the results have been obtained in a worst-case scenario), we report in Figure 10 a breakdown of the time spent in the initialisation phase vs the simulation phase for the CUBA benchmark on the cluster of m5.8xlarge virtual machines. As it can be seen, the simulation pays a non-negligible almost-constant time spent in the initialisation phase, while the simulation phase is scaling significantly. This indicates that if more effective initialisation strategies are devised, our approach might also improve performance when simulating smaller-scale networks.

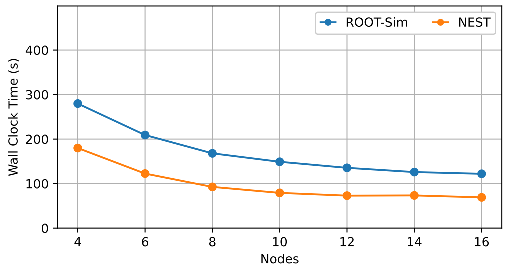


(a) CUBA.

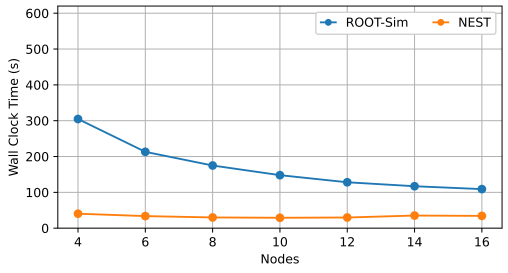


(b) Potjans and Diesmann's.

Figure 7: Distributed Scalability (m5.4xlarge).



(a) CUBA.

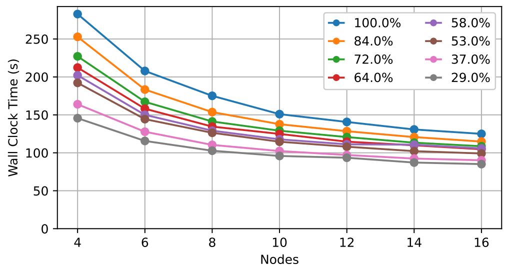


(b) Potjans and Diesmann's.

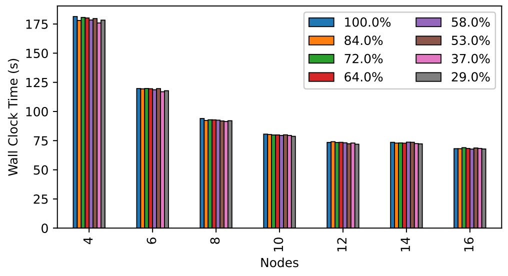
Figure 8: Distributed Scalability (m5.8xlarge).

6 CONCLUSIONS AND FUTURE WORK

We have presented a methodology and technical support to run SNN models on top of Time Warp-based PDES runtime environments. The main results of our contribution are related to the capability



(a) ROOT-Sim.



(b) NEST.

Figure 9: Performance when Varying Network Activity (m5.8xlarge).

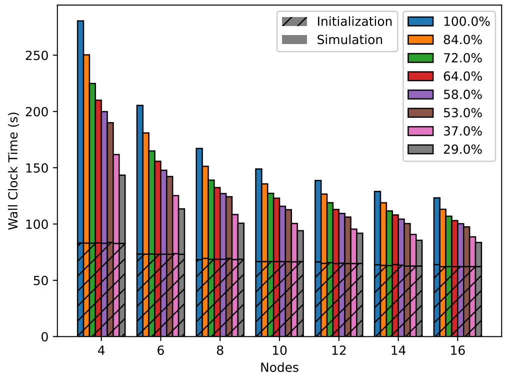


Figure 10: Initialisation vs Simulation Time (m5.8xlarge).

of the PDES simulations to deliver more accurate results with respect to state-of-the-art SNN simulators while providing scalability trends showing that, if extremely large-scale networks are to be simulated, our approach can outperform the same state-of-the-art runtime environments. At the same time, our experimental assessment has shown that some models' configuration parameters make traditional time stepped simulations outperform PDES simulations run on general-purpose runtime environments. We have also identified a potential bottleneck that plays as a show-stopper to deliver timely simulation results. Investigating how to improve this part of the overall simulation will be dealt with in future work.

REFERENCES

[1] Arnon Amir, Pallab Datta, William P Risk, Andrew S Cassidy, Jeffrey A Kusnitz, Steve K Esser, Alexander Andreopoulos, Theodore M Wong, Myron Flickner, Rodrigo Alvarez-Icaza, Emmett McQuinn, Ben Shaw, Norm Pass, and Dharmendra S Modha. 2013. Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores. In *Proceedings of the The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Piscataway, NJ, USA, 1–10. https://doi.org/10.1109/IJCNN.2013.6707078

[2] Rajagopal Ananthanarayanan, Steven K Esser, Horst D Simon, and Dharmendra S Modha. 2009. The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (Portland, Oregon) *(SC)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/1654059.1654124

[3] Peter D Barnes, 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* (Montréal, Québec, Canada) *(SIGSIM PADS '13)*. ACM, New York, NY, USA, 327–336. https://doi.org/10.1145/2486092.2486134

[4] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. 2014. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in neuroinformatics* 7 (2014), 13. https://doi.org/10.3389/fninf.2013.00048

[5] T Binzegger. 2004. A Quantitative Map of the Circuit of Cat Primary Visual Cortex. *Journal of Neuroscience* 24 (2004), 8441–8453. https://doi.org/10.1523/JNEUROSCI.1400-04.2004

[6] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris, Jr, 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. *Journal of computational neuroscience* 23, 3 (Dec. 2007), 349–398. https://doi.org/10.1007/s10827-007-0038-6

[7] Kristofor D Carlson, Michael Beyeler, Nikil Dutt, and Jeffrey 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)*. IEEE, Piscataway, NJ, USA, 570–577. https://doi.org/10.1109/ASPDAC.2014.6742952

[8] Nicholas T Carnevale and Michael L Hines. 2006. *The NEURON Book*. Cambridge University Press, Cambridge, UK. https://doi.org/10.1017/CBO9780511541612

[9] Christopher D Carothers, Kalyan S Perumalla, and Richard M Fujimoto. 1999. Efficient Optimistic Parallel Simulations Using Reverse Computation. *ACM Transactions on Modeling and Computer Simulation* 9 (1999), 224–253. https://doi.org/10.1145/347823.347828

[10] Andrew S Cassidy, Rodrigo Alvarez-Icaza, Filipp Akopyan, Jun Sawada, John V Arthur, Paul A Merolla, Pallab Datta, Marc Gonzalez Tallada, Brian Taba, Alexander Andreopoulos, Arnon Amir, Steven K Esser, Jeff Kusnitz, Rathinakumar Appuswamy, Chuck Haymes, Bernard Brezzo, Roger Moussalli, Ralph Bellofatto, Christian Baks, Michael Mastro, Kai Schleupen, Charles E Cox, Ken Inoue, Steve Millman, Nabil Imam, Emmett Mcquinn, Yutaka Y Nakamura, Ivan Vo, Chen Guok, Don Nguyen, Scott Lekuch, Sameh Asaad, Daniel Friedman, Bryan L Jackson, Myron D Flickner, William P Risk, Rajit Manohar, and Dharmendra 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)*. IEEE, Piscataway, NJ, USA, 27–38. https://doi.org/10.1109/SC.2014.8

[11] Andrew S Cassidy, Jun Sawada, Paul Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Filipp Akopyan, Bryan L Jackson, and Dharmendra S Modha. 2016. *TrueNorth: A High-Performance, Low-Power Neurosynaptic Processor for Multi-Sensory Perception, Action, and Cognition.* Technical Report. Almaden Research Center, IBM Research.

[12] Kit Cheung, Simon R Schultz, and Wayne Luk. 2016. NeuroFlow: A General Purpose Spiking Neural Network Simulation Platform using Customizable Processors. *Frontiers in neuroscience* 9 (Jan. 2016), 1–15. https://doi.org/10.3389/fnins.2015.00516

[13] Ting-Shuo Chou, Hirak J Kashyap, Jinwei Xing, Stanislav Listopad, Emily L Rounds, Michael Beyeler, Nikil Dutt, and Jeffrey L Krichmar. 2018. CARLsim 4: An Open Source Library for Large Scale, Biologically Detailed Spiking Neural Network Simulation using Heterogeneous Clusters. In *Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Piscataway, NJ, USA, 1–8. https://doi.org/10.1109/IJCNN.2018.8489326

[14] Davide Cingolani, Alessandro Pellegrini, and Francesco Quaglia. 2017. Transparently Mixing Undo Logs and Software Reversibility for State Recovery in Optimistic PDES. *ACM Transactions on Modeling and Computer Simulation* 27, 2 (May 2017), 1–26. https://doi.org/10.1145/3077583

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

[16] Richard M Fujimoto. 1990. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (Oct. 1990), 30–53. https://doi.org/10.1145/84537.84545

[17] Richard M Fujimoto. 1990. Performance of Time Warp Under Synthetic Workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, David Nicol (Ed.). Society for Computer Simulation International, San Diego, CA, USA, 23–28.

[18] Marc-Oliver Gewaltig and Markus Diesmann. 2007. *NEST (NEural Simulation Tool)*. Vol. 2. Scholarpedia, Chapter 4. https://doi.org/10.4249/scholarpedia.1430

[19] Samanwoy Ghosh-Dastidar and Hojjat Adeli. 2009. Spiking neural networks. *International journal of neural systems* 19 (2009), 295–308. https://doi.org/10.1142/S0129065709002002

[20] A Goldberg, John A P Sekar, and Jonathan R Karr. 2020. Exact Parallelization of the Stochastic Simulation Algorithm for Scalable Simulation of Large Biochemical Networks. (2020). arXiv:2005.05295 [q-bio.MN]

[21] Samuel Greengard. 2020. Neuromorphic chips take shape. *Commun. ACM* 63, 8 (July 2020), 9–11. https://doi.org/10.1145/3403960

[22] William Gropp. 2012. MPI 3 and Beyond: Why MPI Is Successful and What Challenges It Faces. In *Recent Advances in the Message Passing Interface*, Jesper Larsson Träff, Siegfried Benkner, and Jack J Dongarra (Eds.). Lecture Notes in Computer Science, Vol. 7490. Springer International Publishing, Berlin Heidelberg, Germany, 1–9. https://doi.org/10.1007/978-3-642-33518-1_1

[23] Alexander Hanuschkin, Susanne Kunkel, Moritz Helias, Abigail Morrison, and Markus Diesmann. 2010. A general and efficient method for incorporating precise spike times in globally time-driven simulations. *Frontiers in neuroinformatics* 4 (Oct. 2010), 1–19. https://doi.org/10.3389/fninf.2010.00113

[24] Suzana Herculano-Houzel. 2012. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proceedings of the National Academy of Sciences of the United States of America* 109, Supplement 1 (June 2012), 10661–10668. https://doi.org/10.1073/pnas.1201895109

[25] Suzana Herculano-Houzel and Jon H Kaas. 2011. Gorilla and Orangutan Brains Conform to the Primate Cellular Scaling Rules: Implications for Human Evolution. *Brain, behavior and evolution* 77 (2011), 33–44. https://doi.org/10.1159/000322729

[26] Roger V Hoang, Devyani Tanna, Laurence C Jayet Bray, Sergiu M Dascalu, and Frederick C Harris. 2013. A novel CPU/GPU simulation environment for large-scale biologically realistic neural modeling. *Frontiers in neuroinformatics* 7 (2013), 10. https://doi.org/10.3389/fninf.2013.00019

[27] David R Jefferson. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7, 3 (July 1985), 404–425. https://doi.org/10.1145/3916.3988

[28] Susanne Kunkel, Maximilian Schmidt, Jochen M Eppler, Hans E Plesser, Gen Masumoto, Jun Igarashi, Shin Ishii, Tomoki Fukai, Abigail Morrison, Markus Diesmann, and Moritz Helias. 2014. Spiking network simulation code for petascale computers. *Frontiers in neuroinformatics* 8 (Oct. 2014), 78. https://doi.org/10.3389/fninf.2014.00078

[29] Henry Markram. 2006. The blue brain project. *Nature reviews. Neuroscience* 7, 2 (Feb. 2006), 153–160. https://doi.org/10.1038/nrn1848

[30] Cyrille Mascart, Gilles Scarella, Patricia Reynaud-Bouret, and Alexandre Muzy. 2021. Scalability of large neural network simulations via activity tracking with time asynchrony and procedural connectivity. (June 2021). https://doi.org/10.1101/2021.06.12.448096

[31] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E Granger, Richard P Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J Curry, Andy R Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), 1–27. https://doi.org/10.7717/peerj-cs.103

[32] Kirill Minkovich, Corey M Thibeault, Michael John O'Brien, Aleksey Nogin, Youngkwan Cho, and Narayan Srinivasa. 2014. HRLSim: a high performance spiking neural network simulator for GPGPU clusters. *IEEE transactions on neural networks and learning systems* 25, 2 (Feb. 2014), 316–331. https://doi.org/10.1109/TNNLS.2013.2276056

[33] Quang Anh Pham Nguyen, Philipp Andelfinger, Wentong Cai, and Alois Knoll. 2019. Transitioning Spiking Neural Network Simulators to Heterogeneous Hardware. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation* (Chicago, IL, USA) *(SIGSIM-PADS)*. ACM, New York, NY, USA, 115–126. https://doi.org/10.1145/3316480.3322893

[34] David M Nicol. 1993. The cost of conservative synchronization in parallel discrete event simulations. *J. ACM* 40, 2 (April 1993), 304–333. https://doi.org/10.1145/151261.151266

[35] Daniele M Papetti, Simone Spolaor, Daniela Besozzi, Paolo Cazzaniga, Marco Antoniotti, and Marco S Nobile. 2020. On the automatic calibration of fully analogical spiking neuromorphic chips. In *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Piscataway, NJ, USA, 1–8. https://doi.org/10.1109/IJCNN48605.2020.9206654

[36] Dejan Pecevski. 2009. PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python. *Frontiers in neuroinformatics* 3 (2009), 15. https://doi.org/10.3389/neuro.11.011.2009

[37] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia. 2012. The ROme OpTimistic Simulator: Core Internals and Programming Model. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS)*. ICST, Brussels, Belgium, 96–98. https://doi.org/10.4108/icst.simutools.2011.245551

[38] Matthew D Pickett, Gilberto Medeiros-Ribeiro, and R Stanley Williams. 2013. A scalable neuristor built with Mott memristors. *Nature materials* 12 (2013), 114–117. https://doi.org/10.1038/nmat3510

[39] Mark Plagge, Christopher D Carothers, Elsa Gonsiorowski, and Neil Mcglohon. 2018. NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures. *ACM Transactions on Modeling and Computer Simulation* 28 (2018), 1–25. https://doi.org/10.1145/3186317

[40] Chi-Sang Poon and Kuan Zhou. 2011. Neuromorphic Silicon Neurons and Large-Scale Neural Networks: Challenges and Opportunities. *Frontiers in neuroscience* 5 (2011), 108. https://doi.org/10.3389/fnins.2011.00108

[41] Tobias C Potjans and Markus Diesmann. 2014. The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. *Cerebral cortex* 24 (2014), 785–806. https://doi.org/10.1093/cercor/bhs358

[42] Bruno R Preiss, Wayne M Loucks, and Ian D Macintyre. 1994. Effects of the Checkpoint Interval on Time and Space in Time Warp. *ACM Transactions on Modeling and Computer Simulation* 4 (1994), 223–253. https://doi.org/10.1145/189443.189444

[43] Francesco Quaglia and Andrea Santoro. 2003. Non-Blocking Checkpointing for Optimistic Parallel Simulation: Description and an Implementation. *IEEE Transactions on Parallel and Distributed Systems* 14 (2003), 593–610.

[44] Sanjiv Shah and Mark Bull. 2006. OpenMP. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing* (Tampa, Florida) *(SC)*. ACM, New York, NY, USA, 13. https://doi.org/10.1145/1188455.1188469

[45] Renan O Shimoura, Nilton L Kamiji, Rodrigo F O Pena, Vinicius L Cordeiro, Cesar C Ceballos, Romaro Cecilia, and Antonio C Roque. 2018. [RE] The Cell-Type Specific Cortical Microcircuit: Relating Structure And Activity In A Full-Scale Spiking Network Model. *Zenodo* 34 (2018), 1537–1557. https://doi.org/10.5281/ZENODO.1244116

[46] Donald L Snyder and Michael I Miller. 2012. *Random Point Processes in Time and Space* (second ed.). Springer, New York, NY, USA. https://doi.org/10.1007/978-1-4612-3166-0

[47] Athul Sripad, Giovanny Sanchez, Mireya Zapata, Vito Pirrone, Taho Dorta, Salvatore Cambria, Albert Marti, Karthikeyan Krishnamourthy, and Jordi Madrenas. 2018. SNAVA—A real-time multi-FPGA multi-model spiking neural network simulation architecture. *Neural networks: the official journal of the International Neural Network Society* 97 (Jan. 2018), 28–45. https://doi.org/10.1016/j.neunet.2017.09.011

[48] Marcel Stimberg, Romain Brette, and Dan F M Goodman. 2019. Brian 2, an intuitive and efficient neural simulator. *eLife* 8, e47314 (Aug. 2019), e47314. https://doi.org/10.7554/eLife.47314

[49] Gianluca Susi, Pilar Garcés, Emanuele Paracone, Alessandro Cristini, Mario Salerno, Fernando Maestú, and Ernesto Pereda. 2021. FNS allows efficient event-driven spiking neural network simulations based on a neuron model supporting spike latency. *Scientific reports* 11, 1 (June 2021), 12160. https://doi.org/10.1038/s41598-021-91513-8

[50] Tim P Vogels and L F Abbott. 2005. Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons. *The Journal of neuroscience: the official journal of the Society for Neuroscience* 25, 46 (Nov. 2005), 10786–10795. https://doi.org/10.1523/JNEUROSCI.3508-05.2005

[51] Runchun M Wang, Chetan S Thakur, and André van Schaik. 2018. An FPGA-Based Massively Parallel Neuromorphic Cortex Simulator. *Frontiers in neuroscience* 12 (April 2018), 213. https://doi.org/10.3389/fnins.2018.00213

[52] Esin Yavuz, James Turner, and Thomas Nowotny. 2016. GeNN: a code generation framework for accelerated brain simulations. *Scientific reports* 6 (Jan. 2016), 18854. https://doi.org/10.1038/srep18854