On the Accuracy and Performance of Spiking Neural Network Simulations

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

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

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

***Abstract*—Spiking Neural Networks (SNNs) are a class of Artificial Neural Networks that show a time behaviour that cannot be computed with single one-shot functions. Therefore, to study their evolution over time, simulations are typically employed. Typical simulation approaches rely on time-stepped simulations, while more recent works have highlighted the opportunity to rely on Parallel Discrete Event Simulation (PDES) for improved accuracy. In particular, Speculative PDES has been shown to be a suitable simulation paradigm to deal with the peculiar temporal domain of SNNs. In this paper, we perform an experimental evaluation of these two different approaches, showing the implications on both simulation performance and accuracy. Our assessment showcases that Parallel Discrete Event Simulation can deliver good scaling on parallel architectures while offering more accurate results.**

***Index Terms*—Spiking Neural Networks, Time-Stepped Simulation, Speculative Parallel Discrete Event Simulation, Performance, Accuracy.**

I. Introduction

Spiking Neural Networks (SNNs) are a particular class of Artificial Neural Networks (ANNs) that have seen increasing interest in the last years [1] due to their expressive capabilities. Indeed, SNNs provide a relevant research tool in multiple domains, such as neuroscience, medicine, artificial intelligence, or psychology, because they mimic biological neural networks with high accuracy. When implemented in hardware, they typically show a remarkably reduced energy footprint. This latter point has given rise to neuromorphic chips [2]–[4], which are regarded as a fundamental step ahead in the chase for a good tradeoff between energy efficiency and performance in massively-parallel computing systems.

Simulations are typically the exclusive approach to studying the behaviour of SNNs because their analytical treatment is only possible for special, simplified cases [5]. SNN simulations are collections 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.

SNNs are therefore more challenging to handle than traditional ANNs. Indeed, beyond the neural and synaptic states, SNNs encode data in a temporal domain known as the *spike train* [6]. The output of an SNN is a set of impulses that encode information via their timing, rather than a set of values computed impulsively as in many other ANNs. Moreover, the interconnection between neurons can be arbitrary, and a single neuron can receive multiple spikes in a reduced simulation time window. This makes SNNs belong to the family of continuous-time tightly-coupled models, which are inherently hard to simulate [7]–[9].

In the literature, the complexity of this family of simulations has been coped with by discretising the continuous time and applying conservative methods for time-stepped simulations [10]–[12]. This approach entails observing the state of the simulation at specific time intervals and determining whether new events should be generated given the observed condition. The benefit of this approach is that many existing conservative methods could be employed to efficiently support this kind of simulation on multiple hardware instances (see, e.g., [13]–[16]).

Nevertheless, this approach has several drawbacks. First, typical SNN simulation algorithms rely on a fixed simulation and integration time-step (typically set on the order of tenths of milliseconds) leading to approximated results [17], to the extent that some spikes could be missed even when using neuron models with linear subthreshold dynamics, i.e., where its state evolves in the absence of emitted spikes. Second, the time-stepped nature of SNN simulation algorithms has favoured the focus on neuron models that are handily computable via some iterative numerical procedure, such as the Euler method. Finally, to improve the accuracy of the results, one should reduce the time step, incurring significant performance penalties. Recent results [18], [19] have shown that employing Parallel Discrete Event Simulation (PDES) methods [20] with optimistic synchronisation [21] allows for improved performance and improved accuracy. Indeed, PDES can efficiently skip time intervals where no interaction among the neurons takes place and can further capture the exact simulation time instant at which a particular neuron might spike. At the same time, an optimistic synchronisation algorithm such as Time Warp [21] can capture the inherent parallelism of SNNs, where groups of neurons may spike at

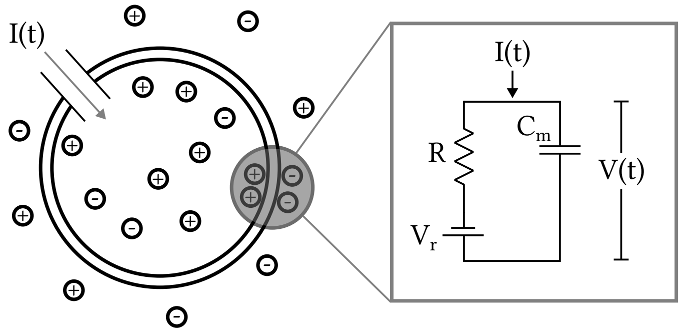

Fig. 1: A neuron is enclosed by the cell membrane (the circle). When it receives a positive input current $I(t)$, it increases the electrical charge inside the cell. The cell membrane acts like a capacitor in parallel with a resistor, which is in line with a battery of potential V_r.

different times.

In this paper, we perform an experimental comparison of these two main simulation techniques. In particular, we focus on state-of-the-art time-stepped simulation algorithms [22] and assess the performance and the accuracy when different time step values are used. We compare these results with a state-of-the-art simulation algorithm for SNN relying on the Time Warp synchronisation algorithm [19]. Our results highlight that if large networks are simulated with high levels of accuracy, the Time Warp synchronisation algorithm can deliver non-negligible performance improvements, enabling more complex scenarios to be feasibly simulated.

The paper is structured as follows. In Section II we introduce some background on SNNs. After an overview of related work in Section III, we describe the considered simulation algorithms for SNNs in Section IV. Section V examines the performance of the reference implementations of these simulation algorithms under different accuracy configurations. The paper concludes with a summary and future work.

II. Background on Spiking Neural Networks

SNNs are based on *spiking neurons*, which communicate by sending signals (spikes) through synapses. Spiking neurons are *stateful*, and the synapses connecting them can be too. 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*.

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. Neurons' plasma membrane isolating properties give rise to a membrane capacitance C_m, and the potential $V(t)$ at time t between the two sides of the membrane is what kick-starts the action potential propagation once it reaches a target threshold value V_{th}. 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 enzyme in charge reverts the neuron to its resting state over a time period, during which the neuron membrane does not charge, called the refractory period τ_{ref}.

Additionally, for the membrane potential to rise, there must be some input current I. Typically, it is the sum of the stimuli $I(t)$ coming at time t from its presynaptic neurons (i.e. those neurons, the outputs of which the neuron receives, as opposed to its postsynaptic neurons, which are the ones that receive the considered neuron's spikes) and some external current I_{ext} that can be supplied (e.g. for experimental observation).

The most commonly-used neuron model in large SNN simulations is the *Leaky Integrate and Fire* (LIF), which is depicted in Figure 1. Equations (1) describe the subthreshold dynamics of the neuron, where $V(t)$ is the membrane potential, and $I(t)$ is the current flowing inside the neuron to the membrane:

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

The positive quantities τ_m and τ_{syn} represent the membrane time constant and the synaptic time constant, respectively. For a more thorough discussion of the meaning of all neuronal parameters, the reader can refer to [6].

Spikes are delivered to post-synaptic neurons with a delay in virtual time[1] and an effect (in terms of delivered potential) established by the synapse model. The majority of PDES simulations employ a synapse model, which is extremely simplistic in its workings, typically referred to as *jump synapse*, characterised by a fixed transmission delay t_{trans} and a weight w. By using this model, a spike causes the post-synaptic neuron to instantaneously increase its $V(t)$ by w.

A more complex synapse model is the *instantaneous raise/ exponential decay synapse*, commonly called just *exponential synapse*. This type of synapse does not directly act on the membrane potential. Rather, it generates an instantaneous increase in the neuron's incoming current. 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 that the neuron might spike in the future: the hypothetical spike time (if any) is computed via numerical methods (if an analytical solution for spike timing is not available for the model), and the resulting event is enqueued.

III. Related Work

The problem of accuracy and performance in SNN simulations is well-known in the literature [17]. In particular,

[1]In a simulation, three different notions of time can be identified [23]: physical time refers to the real-world evolution of the process being simulated; virtual time (or simulation time) refers to the simulated time; wall-clock time refers to the real-world time necessary to run the simulation model.

it directly derives from the availability of multiple methods to solve the neuron model equations and handle spike events [24]–[27]. The technique picked for a particular simulation directly determines the simulation speed and accuracy of the results [28].

To improve accuracy, several works [17], [29]–[31] have employed different numerical approaches or parameter estimation. Overall, resorting to a different numerical approach still depends on the underlying scheduling/synchronisation algorithm. In this sense, the work in [27] tackles the estimation of the error introduced by an SNN simulation. In particular, the authors show that it is possible to separate the numerical integration error from the spike-detection timing error, proper of time stepped simulations.

On the performance side, several works have explored the possibility of running SNN models on various hardware instances, such as GPUs [29], [32], [33], FPGAs [34], [35], or even heterogeneous systems [36]. Again, these works mainly focus on the deployment of time-stepped simulations.

The literature has also considered exploiting different discrete-event simulation algorithms. In [18], the authors have shown that relying on speculative PDES simulation using the Time Warp synchronisation protocol can lead to non-minimal performance improvement when focusing on the TrueNorth Leaky Integrate and Fire (TNLIF) [37] architecture. In [38], discrete-event simulation is used to provide better performance in the case of synapse models showing a spike latency, i.e. a delay exhibited in response to depolarization. The work in [19] is inheriting the approach from [18] and then shows that with proper event management strategies, it is possible to obtain good scalability also when using more complex instantaneous raise/exponential decay synapses.

IV. Simulation Algorithms for SNN

This section illustrates the inner workings of the two SNN simulation algorithms we consider in our experimental assessment.

A. Time Stepped Simulations

As mentioned, the largest part of SNN simulations as, e.g., implemented by the well-known NEST [22] and Brian [39] simulators, rely on time-stepped algorithms, whose high-level pseudocode is provided in Algorithm 1.

This kind of simulation approach is simple. Indeed, all neuron state updates are evaluated periodically by processing the incoming spikes. These spikes increase membrane potential, which is again evaluated numerically in the interval $\mathrm{d}t$. After updating all neurons' states, the simulation algorithm checks which of them (if any) have a membrane potential V_m that has reached the spiking threshold. If this is the case, spikes are sent from each of the ready-to-spike neurons to the respective postsynaptic neurons. To account for synapse delays, the typical strategy is to rely on some sort of future event queue, typically implemented as a circular array [6], that allows keeping track of what spike should be delivered to what neuron at what time(step) in the future.

Algorithm 1: Time Stepped Simulation Algorithm.

```
1  $t = 0$
2  while $t < t_{end}$ do
3      foreach neuron do
4          process incoming spikes
5          advance neuron dynamics by $\mathrm{d}t$
6      foreach neuron do
7          if $V(t) > V_{th}$ then
8              reset neuron membrane
9              foreach connection do
10                 send spike
11     $t \leftarrow t + \mathrm{d}t$
```

The time complexity of this simulation algorithm can be easily computed. The first inner loop accounts for neuron state updates. If there are n neurons in the network, the loop has an $O(n)$ cost. Considering that the physical time of the simulation is divided into intervals of the same size, the cost is $O(n/\,\mathrm{d}t)$ per unit of physical time. Concerning the second inner loop, if we call f the average firing rate of neurons per physical-time unit, assuming that on average each neuron is connected to s other neurons, the cost is $O(fns)$. Under general assumptions, it cannot be stated which of the two components impacts the overall cost more. We can therefore conclude that the cost of this algorithm *per physical-time unit* is:

$$O\left(\frac{n}{\mathrm{d}t} + fns\right). \tag{2}$$

In this computation, we have assumed that activities related to the computation of neuron dynamics and spike delivery are negligible, although depending on the specific used neuron model and the complexity of the topology, it might not always be the case. Anyhow, Equation (2) indicates that the overall cost of the time-stepped simulation grows with the network's size and the simulation's precision, which is exactly one of the key points we assess experimentally in this paper.

An additional issue with the time-stepped simulation algorithm is that spike timings are aligned to a grid defined by the time steps. Therefore, the final result approximates the actual behaviour of the network, even when the numerical methods used to compute differential equations provide exact results. Similarly, since the check on the threshold is carried out only at the time steps (see line 7 in Algorithm 1), some spikes may be missed. This is the second key point that we assess experimentally in this paper.

B. Speculative Discrete Event Simulations

In a discrete-event simulation adhering to the Time Warp synchronisation protocol [21], the simulation model is partitioned into different Logical Processes (LPs), which maintain a portion of the global simulation state with no overlap across the different LPs. According to the simulation algorithm described in [19], each neuron is mapped to a single LP.

Spikes are represented by messages, which are delivered to the destination LP. Since a single neuron can be connected to

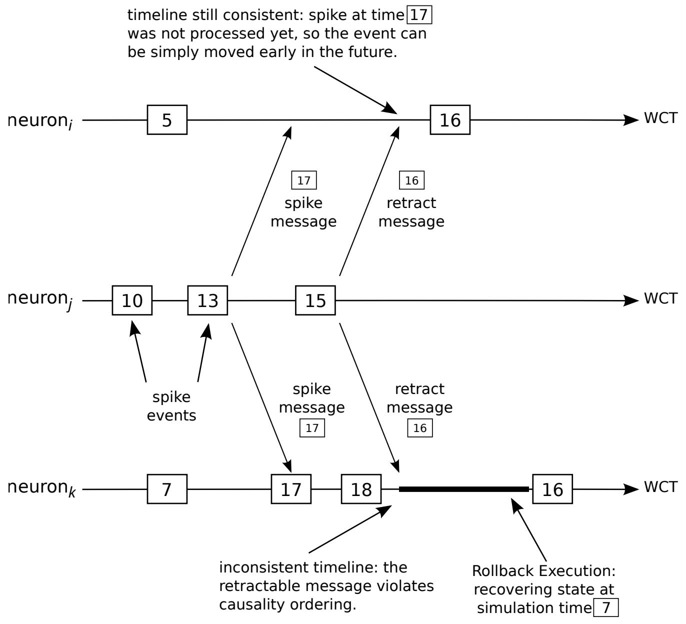
timeline still consistent: spike at time 17 was not processed yet, so the event can be simply moved early in the future.
neuron$_i$
5
16
WCT
17 spike message
16 retract message
neuron$_j$
10
13
15
WCT
spike events
spike message 17
retract message 16
neuron$_k$
7
17
18
16
WCT
inconsistent timeline: the retractable message violates causality ordering.
Rollback Execution: recovering state at simulation time 7

Fig. 2: Retractable Spikes Naïve Scheme. A neuron can decide to change the time of a spike already injected in the system. Receiving neurons might have to rollback part of their execution.

multiple neurons, injecting one spike event for each destination LP could easily thrash the simulation due to significant time spent on event management. Therefore, cross-neuron communication is supported by a form of publish/subscribe events: a spiking neuron will inject a single instance of the spiking event into the system. All destination neurons will subscribe to the events generated by the source one, and the underlying runtime environment will deliver *a copy* of the spiking event, thus significantly reducing the burden on the messaging subsystem.

According to the traditional Time Warp protocol, events are executed independently of their safety. It means that a destination neuron could receive a spike after the simulation time at which it had to be processed. In this case, the state of the neuron is rolled back to a previous time instant, and execution is resumed from a consistent snapshot. During rollback execution, inconsistently-generated spikes are undone by generating so-called antimessages. An antimessage reception could cause additional cascading rollbacks.

Given the nature of the spikes, it is impossible to consistently predict the spiking time given the current state of the neuron. Indeed, a more accurate spiking time could be determined after the neuron receives an upcoming spike. A simple solution at the model level could be to inject in the system *tentative* spikes, i.e. events that could be associated with some per-neuron epoch counter. Every time the neuron state is updated due to the receipt of an incoming spike, the new spiking time could be re-computed. A new spiking event (superseding the previous one) could be injected into the system. Anyhow, this naïve approach is unlikely to scale due to the large amount of extremely-fine grained simulation events that are typically the cause of poor performance in Time Warp simulations [40]. The performance degradation of this scheme stems from the strict decoupling between the model and the runtime environment in Time Warp simulations. In this scenario, the model cannot inform the runtime environment that a tentative spiking event should be removed from the system, and the model can only logically discard it once it is delivered for execution.

For this reason, in [19] the authors introduce the concept of *retractable events*, i.e. events that can be marked by the model as *tentative*. Logically speaking, this support allows implementing tentative spike events, according to the scheme depicted in Figure 2. After a neuron has sent its spikes, a new event could update the time of said spikes. This is the case, e.g., of new spikes being delivered to the neuron, which consequently charges faster, thus reaching the threshold V_{th} earlier. In this case, the neuron can inform the receiving neurons that the spike should be dealt with earlier. At the destination neurons, if the involved spike has not been processed yet, the event is simply moved earlier in the future. Conversely, if it has already been processed, a traditional rollback operation will restore the neuron state to a consistent timestamp, and the new spiking time will be considered in the simulation.

The problem with this naïve approach is that the total number of rollbacks can still be high. Therefore, a straightforward optimisation is to deal with retractable events locally at a single neuron. A neuron locally determines its next firing time and schedules to itself a *tentative spike-firing event*. This tentative firing event is managed as a regular firing event (i.e., the neuron sends the spikes to all destination neurons upon receiving it) if no change in the firing time occurs. Conversely, if the neuron model determines a new timestamp for the firing event, the runtime environment will accordingly act on the message queue. In particular, if the firing event is not yet processed, it will simply be moved to the appropriate new firing time. In this way, the number of events injected into the system and the total number of rollbacks are significantly reduced, as the destination LPs will only receive a spike at the accurate firing time, after that the firing neuron has correctly received all pre-synaptic stimuli.

V. Experimental Assessment

A. Reference Implementations

We rely on two reference implementations for the simulation algorithms described in Section IV[2].

Concerning time-stepped simulations, we rely on the NEST simulator [22]. NEST comes prepacked with “over 50 neuron models, many of which have been published” and “over ten 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

[2]The source code used in the experimentation is available in the reproducibility package at https://doi.org/10.5281/zenodo.6616651.

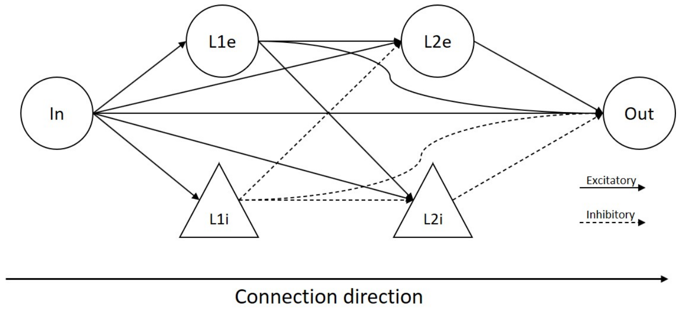

Fig. 3: Schema of the Synthetic Model.

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.

The speculative discrete-event simulation algorithm has been implemented within the ROme OpTimistic Simulator (ROOT-Sim) [41]. ROOT-Sim is an HPC simulation library targeting optimistic simulation on massively parallel multicore machines and distributed compute clusters/supercomputers. All the facilities related to publish/subscribe events and retractable events have been implemented in the simulation library[3]

B. Experimental Setup

The performance experiments were run on an AWS m5.8xlarge machine with 32 vCPUs. These machines are based on Intel Xeon® Platinum 8175M processors, running Ubuntu 20.04.3 LTS, on kernel version 5.13.0-1025-aws. Each experiment was run with 32, 24, 16, 8, and 4 worker threads. NEST only has data points for 16 or more worker threads due to a limitation not allowing more than 2^{27} synaptic connections per worker thread; as such, only ROOT-Sim was run on 4 and 8 workers.

The standard benchmark we have used to run the performance experiments is inspired by a study on signal propagation in linear integrate-and-fire (LIF) models [42]. 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%. The CUBA model is simulated for 10 seconds of simulation time with each simulator while varying the simulation precision. In NEST, this is achieved by selecting a resolution value. In ROOT-Sim, the time tolerance is currently built into the model and can be selected deliberately, as long as the hardware constraints allow it.

For accuracy experiments, we have built a synthetic network model consisting of 1,000 neurons. The network is acyclic,

[3]The source code of the ROOT-Sim library is available at https://github.com/ROOT-Sim/core.

TABLE I: Neurons and populations parameter specification.

Populations and inputs

Name	Input	L1e	L1i	L2e	L2i	Output
Population size	100	200	200	200	200	100

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

TABLE II: Connectivity map for the generated topology.

		to					
		In	L1e	L1i	L2e	L2i	Out
from	In	-	0.292	0.192	0.049	0.237	0.169
	L1e	-	-	-	0.106	0.254	0.438
	L1i	-	-	-	0.409	0.250	0.309
	L2e	-	-	-	-	-	0.491
	L2i	-	-	-	-	-	0.225

TABLE III: Synaptic parameter specification.

Name	Value	Description
w_{exc}	200 pA	Excitatory synaptic strengths
w_{inh}	-600pA	Inhibitory synaptic strength
d_e	1.5 ms	Excitatory synaptic transmission delays
d_i	0.8 ms	Inhibitory synaptic transmission delays

divided into four layers: Input, L1, L2, and Output. A network topology scheme is found in Figure 3. The Input layer comprises 100 excitatory neurons, which receive a constant input current of $1800pA$. Layers L1 and L2 both comprise two populations of 100 excitatory (L1e/L2e) and 100 inhibitory neurons (L1i/L2i). The Output layer consists of 100 neurons. The output neurons' spikes are observed and compared with the ground truth to determine simulator accuracy. Synapses all have fixed weights of $200pA$ with a delay of $1.5ms$ when excitatory and a weight of $-600pA$ and delay of $0.8ms$ when inhibitory.

The network topology and relevant parameters (initial membrane potential, input current, synaptic weight, synaptic delay) are generated with a script into a configuration file, which then is loaded by the models of each simulator, as well as by the script that computes the ground truth, leading to the exact same topology and initial conditions for every single neuron in all three cases.

C. Computing the Ground Truth

As noted earlier, the network chosen for the accuracy evaluation is acyclic, and, conveniently, there is a simple algorithm able to compute its behaviour. Given such an acyclic network, we compute a topological order of the neurons $n_0, n_1, ...n_k$; then, necessarily, the behaviour of a neuron n_i will only depend on the behaviour of neurons $n_0, n_1, ..., n_{i-1}$. That implies that once a simulation time limit t has been selected, it

TABLE IV: Spiking Times for ROOT-Sim and NEST (timestep/error: 0.1). For each result, we provide the spike time (ms) and the neuron number in brackets. The results relate to the first 10 ms of simulated time.

Spike No.	Ground Truth	ROOT-Sim	NEST
1	2.999 (900)	3.046 (900)	3.200 (900)
2	3.556 (977)	3.615 (977)	3.900 (975)
3	3.598 (975)	3.630 (975)	3.900 (950)
4	3.787 (970)	3.771 (970)	6.200 (912)
5	5.843 (953)	5.955 (953)	6.300 (952)
6	—	6.215 (927)	—
7	—	6.667 (923)	—

TABLE V: Spiking Times for ROOT-Sim and NEST (timestep/ error: 0.001). For each result, we provide the spike time (ms) and the neuron number in brackets. The results relate to the first 10 ms of simulated time.

Spike No.	Ground Truth	ROOT-Sim	NEST
1	2.999 (900)	2.999 (900)	3.110 (900)
2	3.556 (977)	3.556 (977)	3.795 (975)
3	3.598 (975)	3.598 (975)	6.486 (952)
4	3.787 (970)	3.787 (970)	—
5	5.843 (953)	5.842 (953)	—

is possible to simulate the neurons one by one, starting from n_0 through n_k feeding the output from the neurons to the correct post-synaptic ones. Since there is no analytical closed-form solution for the spike times for the LIF neuron used in this network, we still have to resort to numerical methods. We are not concerned with performance in this case. Therefore, we implemented a Python script carrying out the described computations with an error factor of $10^{-9}ms$.

D. Accuracy and Performance Results

We report in Tables IV and V the results obtained running the synthetic model on ROOT-Sim and NEST, compared to the ground truth results obtained according to the method described in Section V-C. We have set the timestep/accuracy factor to 0.1 (Table IV) and 0.001 (Table V). The results report the spikes obtained in the simulation's first 10 ms. We provide the spiking time and the ID of the neuron that generated the spike in the Output layer for each spike.

By the results in Table IV, we observe that, for both simulators, the accuracy is not high. In particular, ROOT-Sim generates spikes at the correct neurons, but the difference in spiking times is between 1% and 2%, in a significantly reduced simulation time. Interestingly, this model generates two additional spurious spikes. Conversely, NEST has a higher error (up to 60%), but more interestingly, it induces spikes at the wrong neurons, except for the first one. The number of spikes is anyhow correct. These results are expected. Indeed, given the nature of the synthetic model, it is clear that a low resolution is unlikely to provide accurate results due to the strong interaction between excitatory and inhibitory neurons.

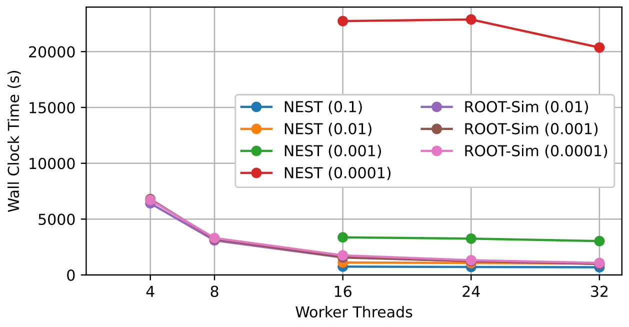

Fig. 4: Performance Comparison.

The results with a higher resolution, provided in Table V, show that the results based on ROOT-Sim deliver much higher accuracy. Conversely, NEST results show that two spikes are missing, spikes are induced at the wrong neurons, and the accuracy is still low (with an error ranging from 3.7% to 80%). One could wonder how the two examined simulators may deliver a different accuracy even when using the same value for the time-step/error. As mentioned in Section IV, the sources of inaccuracy are essentially two. Common to both algorithms, the first one is due to computational inaccuracy in spike timings: small deviations can cause post-synaptic neurons to emit or miss a spike when they should not have. The second source of accuracy loss is specific to NEST only, and it is due to how the spike detection works. With the default settings, a spike is detected only if the firing threshold potential is overcome at one discrete time-step. In other words, NEST assumes that a neuron can never overcome the firing threshold if it has not done so at the beginning and the end of the time step, which can lead to missing a spike in some edge cases, even with a single neuron.

As for performance evaluation, we can refer to Figure 4, where both simulators used the standard current-based (CUBA) synaptic interactions benchmark [6] to simulate 10 seconds of physical time, with varying degrees of accuracy. NEST only has data for 16 or more workers because it does not allow for more than 2^{27} synaptic connections per worker thread. While NEST outperforms ROOT-Sim in terms of speed for low-resolution values (10^{-1} and 10^{-2}), when running with a resolution of 10^{-3}, the performance dramatically degrades, leaving the edge to ROOT-Sim, even when the latter runs on eight workers. With a resolution of 10^{-4}, the NEST time-to-solution is significantly larger, with the best configuration (using 32 worker threads), taking over $20,369$ seconds to complete, while ROOT-Sim took $1,076$—this is 18,92x. This result is expected, as multiplying the resolution tenfold also multiplies the number of calculations needed. It is not unreasonable to expect higher resolutions to be practically unfeasible for sizeable networks.

Increasing simulation resolution appears to have a minimal impact on ROOT-Sim's performance, allowing it to be increased almost at will without the risk of running into prohibitive time costs.

VI. Conclusions

In this paper, we have presented an experimental assessment of the tradeoff between the performance and accuracy of two algorithms to simulate SNNs. By the results, it is clear that if a high resolution in the results is pursued, traditional time-stepped simulation algorithms cannot provide results promptly, while speculative PDES-based ones exhibit only a reduced performance penalty. At the same time, also with lower time-step values, although the performance of the time-stepped simulation outperforms the speculative PDES' one, the accuracy of time-stepped algorithms still appears to be lower.

While the considered speculative PDES-based algorithm allows for the execution of non-trivial models, e.g., based on exponential synapses, future work will entail implementing and studying multiple neuron/synapse model to study the effect on both accuracy and performance.

References

[1] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, pp. 295–308, 2009.

[2] S. Greengard, "Neuromorphic chips take shape," *Communications of the ACM*, vol. 63, no. 8, pp. 9–11, Jul. 2020.

[3] M. D. Pickett, G. Medeiros-Ribeiro, and R. S. Williams, "A scalable neuristor built with mott memristors," *Nature materials*, vol. 12, pp. 114–117, 2013.

[4] C.-S. Poon and K. Zhou, "Neuromorphic silicon neurons and Large-Scale neural networks: Challenges and opportunities," *Frontiers in neuroscience*, vol. 5, p. 108, 2011.

[5] N. Brunel, "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons," *Journal of computational neuroscience*, vol. 8, no. 3, pp. 183–208, May 2000.

[6] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, Jr, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. El Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: a review of tools and strategies," *Journal of computational neuroscience*, vol. 23, no. 3, pp. 349–398, Dec. 2007.

[7] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, 1987.

[8] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The journal of physical chemistry. A*, vol. 104, no. 9, pp. 1876–1889, Mar. 2000.

[9] P. Andelfinger and A. Uhrmacher, "Optimistic parallel simulation of tightly coupled agents in continuous time," in *Proceedings of the 2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications*, ser. DS-RT. Piscataway, NJ, USA: IEEE, Sep. 2021, pp. 1–9.

[10] J. Kent Peacock, J. W. Wong, and E. G. Manning, "Distributed simulation using a network of processors," *Computer Networks (1976)*, vol. 3, no. 1, pp. 44–56, Feb. 1979.

[11] D. M. Nicol, C. C. Micheal, and P. Inouye, "Efficient aggregation of multiple LPs in distributed memory parallel simulations," in *Proceedings of the 1988 Winter Simulation Conference*, ser. WSC, E. A. MacNair, K. J. Musselman, and P. Heidelberg, Eds. Piscataway, NJ, USA: IEEE, 1989.

[12] A. Ferscha and S. K. Tripathi, "Parallel and distributed simulation of discrete event systems," University of Maryland at College Park, Tech. Rep., 1994.

[13] S. C. Tay, G. S. H. Tan, and K. Shenoy, "Algorithms and analyses: piggy-backed time-stepped simulation with 'super-stepping'," in *Proceedings of the 2003 Winter Simulation Conference*, ser. WSC, S. E. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, Eds. Catonsville, MD, USA: Informs, 2003, pp. 1077–1085.

[14] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, "FLAME: simulating large populations of agents on parallel hardware architectures," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS. Richland, SC, USA: IFAAMAS, 2010, pp. 1633–1636.

[15] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, and C. Spagnuolo, "Bringing together efficiency and effectiveness in distributed simulations: The experience with D-Mason," *Simulation*, vol. 89, no. 10, pp. 1236–1253, Oct. 2013.

[16] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A survey on agent-based simulation using hardware accelerators," *ACM Computing Surveys*, vol. 51, pp. 1–35, 2019.

[17] A. Hanuschkin, S. Kunkel, M. Helias, A. Morrison, and M. Diesmann, "A general and efficient method for incorporating precise spike times in globally time-driven simulations," *Frontiers in neuroinformatics*, vol. 4, pp. 1–19, Oct. 2010.

[18] M. Plagge, C. D. Carothers, E. Gonsiorowski, and N. Mcglohon, "NeMo: A massively parallel Discrete-Event simulation model for neuromorphic architectures," *ACM Transactions on Modeling and Computer Simulation*, vol. 28, pp. 1–25, 2018.

[19] A. Pimpini, A. Piccione, B. Ciciani, and A. Pellegrini, "Speculative distributed simulation of very large spiking neural networks," in *Proceedings of the 2022 SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS. New York, NY, USA: ACM, 2022.

[20] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[21] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[22] M.-O. Gewaltig and M. Diesmann, *NEST (NEural Simulation Tool)*. Scholarpedia, 2007, vol. 2, ch. 4.

[23] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*. Wiley, Jan. 2000.

[24] D. F. M. Goodman, "The brian simulator," *Frontiers in neuroscience*, vol. 3, pp. 192–197, 2009.

[25] M. Mattia and P. Del Giudice, "Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses," *Neural computation*, vol. 12, no. 10, pp. 2305–2329, Oct. 2000.

[26] M. Lundqvist, M. Rehn, M. Djurfeldt, and A. Lansner, "Attractor dynamics in a modular network model of neocortex," *Network*, vol. 17, no. 3, pp. 253–276, Sep. 2006.

[27] S. Henker, J. Partzsch, and R. Schüffny, "Accuracy evaluation of numerical methods used in state-of-the-art simulators for spiking neural networks," *Journal of computational neuroscience*, vol. 32, no. 2, pp. 309–326, Apr. 2012.

[28] A. Morrison, S. Straube, H. E. Plesser, and M. Diesmann, "Exact subthreshold integration with continuous spike times in discrete-time neural network simulations," *Neural computation*, vol. 19, no. 1, pp. 47–79, Jan. 2007.

[29] D. Yudanov, M. Shaaban, R. Melton, and L. Reznik, "GPU-based simulation of spiking neural networks with real-time performance & high accuracy," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2010, pp. 1–8.

[30] C. Mascart, G. Scarella, P. Reynaud-Bouret, and A. Muzy, "Scalability of large neural network simulations via activity tracking with time asynchrony and procedural connectivity," Jun. 2021.

[31] D. M. Papetti, S. Spolaor, D. Besozzi, P. Cazzaniga, M. Antoniotti, and M. S. Nobile, "On the automatic calibration of fully analogical spiking neuromorphic chips," in *Proceedings of the 2020 International Joint Conference on Neural Networks*, ser. IJCNN. Piscataway, NJ, USA: IEEE, Jul. 2020, pp. 1–8.

[32] M. A. Bhuiyan, V. K. Pallipuram, M. C. Smith, T. Taha, and R. Jalasutram, "Acceleration of spiking neural networks in emerging multi-core and GPU architectures," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8.

[33] J.-P. Tiesel and A. S. Maida, "Using parallel GPU architecture for simulation of planar I/F networks," in *2009 International Joint Conference on Neural Networks*, Jun. 2009, pp. 3118–3123.

[34] D. Thomas and W. Luk, "FPGA accelerated simulation of biologically plausible spiking neural networks," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. ieeexplore.ieee.org, Apr. 2009, pp. 45–52.

[35] L. P. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1, pp. 13–29, Dec. 2007.

[36] Q. A. P. Nguyen, P. Andelfinger, W. Cai, and A. Knoll, "Transitioning spiking neural network simulators to heterogeneous hardware," in *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS. New York, NY, USA: ACM, May 2019, pp. 115–126.

[37] A. S. Cassidy, J. Sawada, P. Merolla, J. V. Arthur, R. Alvarez-Icaza, F. Akopyan, B. L. Jackson, and D. S. Modha, "TrueNorth: A High-Performance, Low-Power neurosynaptic processor for Multi-Sensory perception, action, and cognition," Almaden Research Center, IBM Research, Tech. Rep., 2016.

[38] G. Susi, P. Garcés, E. Paracone, A. Cristini, M. Salerno, F. Maestú, and E. Pereda, "FNS allows efficient event-driven spiking neural network simulations based on a neuron model supporting spike latency," *Scientific reports*, vol. 11, no. 1, p. 12160, Jun. 2021.

[39] M. Stimberg, R. Brette, and D. F. M. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, no. e47314, p. e47314, Aug. 2019.

[40] R. M. Fujimoto, "Performance of time warp under synthetic workloads," in *Proceedings of the SCS Multiconference on Distributed Simulation*, D. Nicol, Ed. San Diego, CA, USA: Society for Computer Simulation International, 1990, pp. 23–28.

[41] A. Pellegrini, R. Vitali, and F. Quaglia, "The ROme OpTimistic simulator: Core internals and programming model," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS. Brussels, Belgium: ICST, Apr. 2012, pp. 96–98.

[42] T. P. Vogels and L. F. Abbott, "Signal propagation and logic gating in networks of Integrate-and-Fire neurons," *The Journal of neuroscience: the official journal of the Society for Neuroscience*, vol. 25, no. 46, pp. 10 786–10 795, Nov. 2005.