# Simulation of Large Spiking Neural Networks on Distributed Architectures, The "DAMNED" Simulator

Anthony Mouraud and Didier Puzenat

GRIMAAG laboratory, Université Antilles-Guyane, Pointe-à-Pitre, Guadeloupe
{amouraud,dpuzenat}@univ-ag.fr

**Abstract.** This paper presents a spiking neural network simulator suitable for biologically plausible large neural networks, named DAMNED for "Distributed And Multi-threaded Neural Event-Driven". The simulator is designed to run efficiently on a variety of hardware. DAMNED makes use of multi-threaded programming and non-blocking communications in order to optimize communications and computations overlap. This paper details the even-driven architecture of the simulator. Some original contributions are presented, such as the handling of a distributed virtual clock and an efficient circular event queue taking into account spike propagation delays. DAMNED is evaluated on a cluster of computers for networks from $10^3$ to $10^5$ neurons. Simulation and network creation speedups are presented. Finally, scalability is discussed regarding number of processors, network size and activity of the simulated NN.

## Introduction

Spiking neurons models open new possibilities for the Artificial Neural Networks (ANN) community, especially modelling biological architectures and dynamics with an accurate behaviour [6]. The huge computation cost of some individual neuron models [7, 9] and the interest of simulating large networks (regarding non-spiking ANN) make Spiking Neural Networks (SNN) good candidates for parallel computing [3]. However, if distributed programming has become easier, reaching good accelerations implies a huge work and a specific knowledge. Therefore we present a distributed simulator dedicated to SNN, with a minimal need of (sequential) programming to deal with a new project, eventually no programming at all. Our simulator is named **DAMNED** for **Distributed And Multi-threaded Neural Event-Driven** simulator. It is designed to run efficiently on various architectures as shown section 1, using an "event-driven" approach detailed in section 2. Sections 3 and 4 present two optimisations, optimized queues and a distributed approach of virtual time handling. Section 5 presents the creation of a SNN within DAMNED and section 6 analyses the scalability of DAMNED. Finally section 7 presents some future works.

D. Palmer-Brown et al. (Eds.): EANN 2009, CCIS 43, pp. 359–370, 2009.

## 1 MIMD-DM Architectures

Our goal is to build large SNN, so the parallel architecture must be scalable in computational power. Large SNN need large amount of memory so the architecture must be scalable in memory which is not possible with shared memory architectures. Thus, our simulator is optimized to run on MIMD-DM architectures. MIMD stands for *Multiple Instruction stream, Multiple Data stream* using Flynn's taxonomy [4], and means several processors working potentially independently; DM stands for *Distributed Memory* which means each processor accesses a private memory, thus communications between processors are necessary to access information located in non-local memory. Another advantage of using MIMD-DM architecture is the diversity of production hardware: from the most powerful parallel computers to the cheapest workstation clusters. The drawback of MIMD-DM architecture is the complexity of the program design; the need of a specific knowledge (message passing instead of well known shared variable programming); the complexity of the code (not to mention debugging and validation). However since DAMNED is end-user oriented, the distributed programming work is already done and validated. The user has no need to enter the parallel code or even to understand the design of the simulator.

Development has been done in `C++` using the **MPI 2.0** library [5]. MPI stands for *Message Passing Interface* and provides methods to handle communications (blocking, non-blocking, and collectives) and synchronization. MPI can launch several tasks (the name of a process in MPI) on a single processor thanks to the scheduler of the host operating system. In such a case communications will not involve the physical network when occurring between two tasks hosted by the same processor. As a consequence, DAMNED runs out of the box on a SMP architecture or on a hyper-threaded architecture. Thereby DAMNED has been developed and tested on a single SMP dual-core ultra-portable laptop computer and validated on a MIMD-DM cluster of 35 dual-core workstations.

The DAMNED simulator is available free of charge; most implementations of MPI are free; the most suitable host operating system is Linux; and the target architecture can be a non-dedicated PC cluster with a standard ethernet network. As a consequence installing and running DAMNED potentially only cost electricity if it is used at time machines were used to be swhiched off.

## 2 Architecture of DAMNED

Simulating SNN implies dealing with time [1]. The simulator can divide the biological time in discrete steps and scan all the neurons and synapses of the network at each time step, this method is named "clock-driven". Our simulator uses another strategy named "event-driven" [17]. An event is a spike emission from pre-synaptic neurons towards post-synaptic neurons, and each event is stamped with the spike emission date. Thus, DAMNED is based on an infinite loop processing events. Each time an event is processed, the simulator (i) actualizes the state of the impacted neuron which eventually generates new events

for post-synaptic neurons; and (ii) increases its virtual clock to the event time stamp. The event-driven strategy is suitable for spiking neuron simulation [12]. Indeed biological spike flows are generally irregular in time with a low average activity. Such irregular activities imply high and low activity periods. While a high activity period benefits to the clock-driven approach, all low activity periods advantage the event-driven approach. Furthermore, a clock-driven simulation looses the order of emission of spikes emitted at the same time step which can change the behaviour of the simulation [15]. In an event-driven simulation, temporal precision only depends on the precision of the variable used for time stamps and clocks, and even a simple 16 bits variable gives a suitable precision.

However, an event-driven simulation does have some drawbacks. The state of neurons is only actualized when events are processed, on reception of a spike on a target neuron, then this neuron decides to spike or not. However when the behaviour of the neuron is described by several differential equations and/or when synaptic impacts are not instantaneous. It is possible to make a prediction of the spike emission date [10]. Such a prediction implies heavy computational costs, and a mechanism must control *a posteriori* the correctness of the prediction.

To efficiently run variety of experiments, DAMNED relies on a single "**front-end task**" (typically running on a workstation) and on several "**simulation tasks**" (typically running on a cluster). The front-end sends inputs into the simulated spiking network and eventually receives outputs. The input can be a flow of stimuli, for example matching a biological experiment. In such a case the evolution of neurons can be monitored to compare the behaviour of the simulated network with biological recordings. The input can also be a physical device such as a camera, in such an example the output could be used to control a motor moving the camera, eventually producing new inputs. More generally, the front-end can host a virtual environment producing inputs for the SNN and eventually being modified by the output of the SNN.

The simulation tasks run the heart of the simulator, that is the processing of events; each task holds a part of the whole neural network. In a simulation of $N$ neurons with $P$ tasks, each task handles $N/P$ neurons. A simulation task is composed of two threads named CMC and CPC, respectively for "**ComMunication Controller**" and "**ComPutation Controller**". The use of threads takes advantage of hyper-threaded and dual-core processors. More precisely, CPC and CMC share two priority queues to manage two types of events:

- incoming Events to be Processed (by local neurons) are in a so called "EtoP" queue, an EtoP is the result of an incoming spike and contains the target neuron ID, the source neuron ID, and the spike emission date;
- outgoing Events to be Emit are in a so called "EtoE" queue, an EtoE is the result of a local neuron spiking and contains the source neuron ID, the emission date, and a boolean flag used to eventually invalidate a predicted emission (see second paragraph of current section).

The simulation is an infinite loop where CPC processes the top event of the EtoP queue (step "CPC 1" on figure 1). The targeted neuron is activated and eventually spikes which generates an EtoE stored in the EtoE ordered queue

according to the event emission date (step "CPC 2"). The "activation" of the target neuron implies changing its state according to its own dynamics. This processing can be achieved by CPC or by a dedicated thread as shown figure 1. Also within an infinite loop, CMC processes the top event of the EtoE queue (step "CMC 2"). CMC knows the tables of postsynaptic neurons ID for all its neurons so the processed EtoE is used to generate EtoPs in the EtoP queue (spikes emitted to local neurons) and to pack EtoPs in messages to be sent to other simulation tasks (spikes emitted to remote neurons). Some controls are performed by CMC and CPC, respectively "EC" and "PC" on figure 1, to keep the behaviour strictly conservative and avoid deadlocks (see section 4).

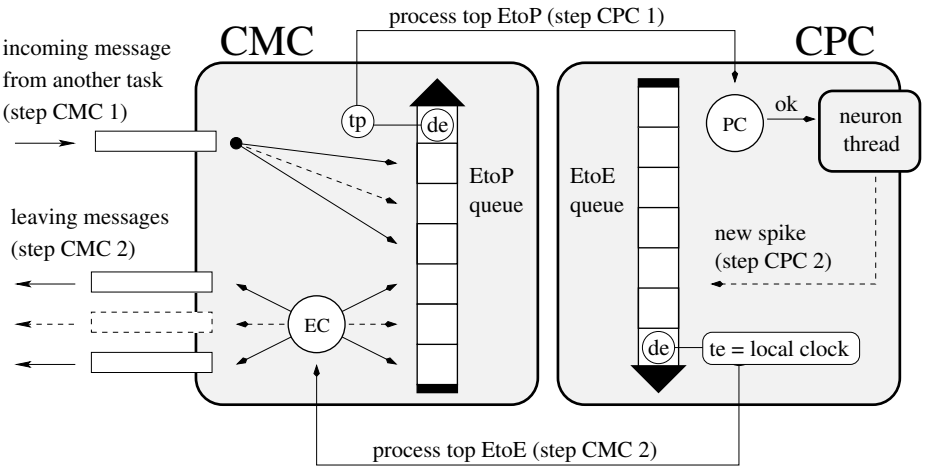


**Fig. 1.** The "ComMunication Controller" and "ComPutation Controller" threads

## 3 Delayed Queues of Events

In a first implementation, CMC and CPC were using priority queues from the `C++` Standard Library (STL), thus accessing and deleting the top event had a null computational cost, but the insertion of an event had an $O(n \times \log(n))$ cost where $n$ is the actual size of the queue. Such a cost becomes significant when queues contain tens of thousands of events, which can be an issue for DAMNED scalability.

To address this potential scalability issue, we have developed "**delayed queues**" (inspired from "calendar queues" [13]) taking into account the travel time of real (*i.e.* biological) spikes between source and target neurons (figure 2). Such a queue is based on two ordering criteria: (i) the time stamp of the event and (ii) the excitatory or inhibitory nature of the spike. Indeed, all inhibitory events for a given time increment has to be computed before excitatory ones to avoid the emission of erroneous events. According to biological knowledge [16], we consider that delays are finite and that the maximal delay $\delta_{max}$ is known at the beginning of the simulation. Furthermore we assume a discrete time whatever the scale could be. It is

then possible to define a type of priority queue allowing no additional cost when inserting a new event nor accessing or deleting the next event. This delayed queue is a circular list containing as many sets of events as existing time stamps in the maximal delay $\delta_{max}$. Each new event is inserted at the index corresponding to time $de+\delta$ , where $de$ is the actual time of emission and $\delta$ is the delay between source and target fields of the event. Knowing the maximal delay $\delta_{max}$ ensures that at time $de$ no incoming event could be inserted later that $de + \delta_{max}$. When every event at time $de$ are computed and the authorization to increase actual virtual time has been given (see next section) the delayed queue moves its first index to $de + \delta t$ where $\delta t$ is the authorized time increment.

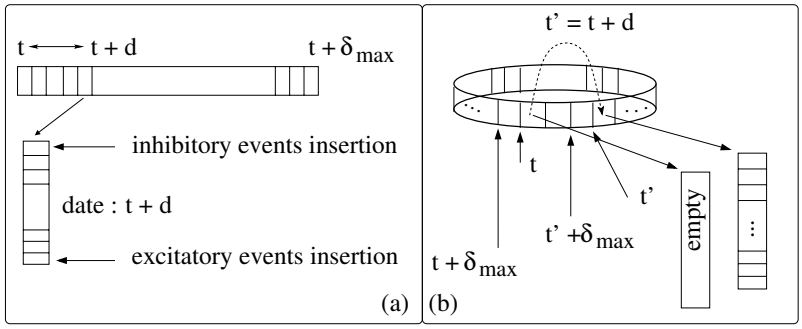


**Fig. 2.** Example of a delayed circular event queue

## 4 Conservative and Distributed Virtual Clock Handling

Communication protocols and MPI ensure that no message is lost. However, the distributed nature of DAMNED implies the risk of a message arriving late since each task runs the simulation at its own speed depending on the activity of local neurons (and eventually of the computer load in a non dedicated cluster). Taking into account such a late message would mean tracking changes made all over the simulation since the time stamp of the late events and restart the simulation from this date, which would be dramatically time consuming. An alternative is to introduce a synchronization protocol. A simple way to keep all tasks synchronous is to implement a centralized clock but this strategy may lead to a bottleneck and would be a limitation for the scalability of the simulator [11]. Therefore we introduce a **distributed clock handling method** [2] based on (i) the exchange of local virtual clocks between simulation tasks, and (ii) the implementation of controls to manage each task virtual clock incrementation making sure a task does not run too fast regarding others.

The local virtual clock on a given task $T_i$ is CMC emission time $te_i$, that is the stamp of the top event of the EtoE queue. From there we introduce a global distributed clock as an array of $te_j$ where $te_j$ ($i \neq$ j) are the last received values of all $T_j$ local clock. Each time a task $T_{src}$ sends an event message to a task $T_{tgt}$,

the whole clock array of $T_{src}$ is sent as a preamble. On $T_{tgt}$, the received $te_i$ ($\forall i \neq tgt$) indicate $T_i$ is granted to run the simulation till date $te_i$. The received clock array is used to actualize the local clock array by keeping the higher date between received and local elements, excepted for the local $te$. The CPC thread also handles its own clock standing for current event processing time and named $tp$, set from the time stamp of the top EtoP queue. Please note that, as described in following paragraphs, DAMNED will set $te$ and $tp$ to null or even negative value to carry special informations.

While no clock change occurs, CPC and CMC continue to respectively process EtoPs and send EtoEs. However when CMC is about to increment $te$ (the task virtual clock), it is necessary to check if the task is not running the simulation too fast regarding other tasks with an "Emission Control" ("EC" on figure 1). This issue also exists for CPC regarding $tp$, thus a "Processed Control" is performed ("PC" on figure 1) when CPC is about to increment $tp$. An inaccurate element of the clock array on one of the task can lead to an erroneous check and blocks CPC or CMC resulting in a deadlock of the simulator. DAMNED uses a boolean array on each task $T_i$ to ensure clock array accuracy: `clock_sent[j]` is set to *true* when the actual value of $te_i$ is sent to task $T_j$ (eventually without events); and each element of `clock_sent` is set to *false* when $te_i$ increases.

When the EtoE queue is empty, CMC updates $te_{src}$ as described by algorithm 1 where $\delta_{min}$ is the minimal "biological" delay within the SNN. The variable $tp$ may has been set to null by CPC (as described in next paragraph), in such a case algorithm 1 enters line 2 if $T_{src}$ has nothing to do, and sending a null clock value will inform other tasks not to wait $T_{src}$. Finally, if EtoE is empty but not EtoP (line 1), CMC sends $-|tp_{src} - \delta_{min}|$ which is the opposite value of the date untill which $T_{src}$ is sure that other tasks can be granted to emit.

```
   if tp_src ≠ 0 then
1  |  te_src ⟵ −|tp_src − δ_min|
   else
2  |  te_src ⟵ 0
```

**Algorithm 1.** Update of the actual virtual time

The control performed by CMC ("EC" on figure 1) on $T_i$ is presented in algorithm 2 where $de$ is the sending date of the processed EtoE. The first test (line 1 of algorithm 2) is a DAMNED mechanism that gives CPC the possibility to generate EtoEs that can be later invalidated, thus an invalidated EtoE treated by CMC is immediately destroyed (line 2) and emitting no more necessary. Such a mechanism is for example used if the emitting date is a prediction (see second paragraph of section 2). More tests are needed if an update of the virtual clock occurs (lines 3 to 6). First, if CPC has been authorized to process an event later than $|de + \delta_{min}|$ (line 3), checking all $te_j$ (line 4) can be avoided which saves an O(P) loop. Otherwise, CMC checks two conditions (line 4): (i) the actual local clock must have been sent to all other tasks and (ii) the EtoE emission date ($de$) must be smaller than the possible processing date of a future received event (if

any, *i.e.* if $te_j \neq 0$). If both conditions are fulfilled local clock can be incremented ($te_i \leftarrow de$) and emission granted (line 5), else emission is denied and $te_i$ is set to $-de$ (line 6) meaning CMC is blocked till a future emission at $de$.

```
1 if EtoE not valid then
2 |  destroy EtoE
  |  → no emission necessary.
  else
  |  if de = te_i then
  |  |  → emission granted.
  |  else
3 |  |  if (tp_i > 0) and (tp_i ≥ de + δ_min) then
  |  |  |  te_i ⟵ de
  |  |  |  → emission granted.
  |  |  else
4 |  |  |  if clock_sent[j] and ((te_j = 0) or
  |  |  |  (de ≤ |te_j| + δ_min)), 0 ≤ j ≤ P − 1 (i ≠ j) then
5 |  |  |  |  te_i ⟵ de
  |  |  |  |  → emission granted
  |  |  |  else
6 |  |  |  |  te_i ⟵ −de
  |  |  |  |  → emission denied
```

**Algorithm 2.** CMC checking an EtoE emission on a task $T_i$

The control performed by CPC before activating the neuron targeted by the top EtoP ("PC" on figure 1) is presented in algorithm 3. CPC is free to process events while (i) $tp_i$ does not have to be incremented (line 1) or (ii) the incrementation keeps $tp_i$ in the range of the lookahead provided by $\delta_{min}$: CPC must check that all tasks have an up-to-date $te_i$ and that it is not any more possible to receive an anterior event that could void the neuron activation (line 2). If these conditions are fulfilled, $tp_i$ can be incremented to $de$ and EtoP can be processed (line 3). Otherwise CPC is blocked until the reception of a sooner event (*i.e.* until time $de$) and $tp_i$ is set to the opposite of $de$ (line 4). Furthermore, if the EtoE queue is empty at processing time (line 3) $te_i$ is set to the opposite of $tp_i$.

```
1 if (tp_i = de) or
2 (clock_sent[j] and (de ≤ |te_j| + δ_min), 0 < j < P) or (te_j = 0)) then
  |  tp_i ⟵ de
3 |  if (te_i = 0) then
  |  |  te_i ⟵ −tp_i
  |  → processing granted
  else
4 |  tp_i ⟵ −de
  |  → processing denied
```

**Algorithm 3.** CPC checking an EtoP processing on a task $T_i$

## 5 Configuration of DAMNED and Definition of a SNN

The definition of a specific network to simulate is done though human friendly text files, with a mark-up language. The user can define "populations of neurons", each population can have a given size and be composed of a given proportion of excitatory or inhibitory neurons. A population can project synapses to other populations, and any type of projection are possible. Of course the density of connexions for a given projection can be chosen, as well as the range of weights and delays to be used. A point of significant interest is that each simulation task creates only its own part of the SNN. Network creation is then also distributed and speeded up (as presented new section).

The user can easily define an environment with input and output cells and stimulation protocols. The environment is a MPI task running on the front-end. It is indeed possible to perform measures of the activity of neurons and to plot classical biological experiments graphs such as "raster" (see figure 3), which facilitate comparisons with biological recordings.

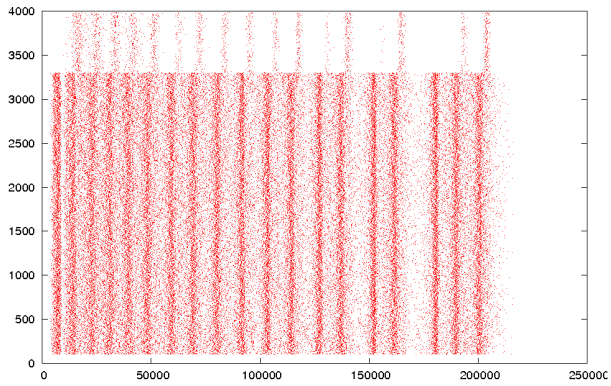


**Fig. 3.** Example of a "raster" output generated from a DAMNED simulation

However, for now the types of neurons used by a network is defined in the code and changing the type used imply to recompile DAMNED. Implemented models include LIF [8], GIF [14], and SRM0 [6]. The implementation of a new model implies writing a new `C++` object but does not need parallel programming. The difficulty deeply depends of the type of neuron, and can be significant, for example if the spiking date must be predicted.

## 6 Results

The DAMNED simulator has been tested on a cluster, that is a classroom of the French West Indies University (*Université des Antilles et de la Guyane*) with 35 intel 2.2 GHz dual-core computers. Each machine has 1 gigabyte of RAM and

is runing Mandriva GNU-Linux (a desktop oriented distribution, not a cluster optimized distribution). The network is based on a 100 Mb/s switch hosting 40 machines. The computers – and the network – were not dedicated for DAMNED so other processes may have influenced negatively the measured performances of DAMNED. However the simulator has been tested at night and all presented results are averaged values based on at least 5 executions making results quite trustful. Of course, DAMNED can run 24 hours a day at a low priority ("niced") without disturbing students, but our present concern is about accurate measures of DAMNED performances which rely on resources as dedicated as possible.

All experiments have been done for a 1 s duration of biological activity. Connectivity of the SNN is set to a biologically plausible 1000 connexions per neuron, meaning that what ever the size of the network each cell is connected to 1 000 other cells. Finally, activity has been tested at 1 Hz and 10 Hz. As an example, for 1000 neurons at 1 Hz, every neuron will spike (only one time) during the 1 s of the simulation, so DAMNED will handle 1000 spikes.

The best way to evaluate the performances of a distributed program is to measure its **speedup** for a given number of processors $P$ as presented in equation 1 where $T_i$ is the execution time on $i$ processor(s). An accurate way to know the "best sequential time" is to develop a perfect simulator optimized for a single processor. However, in the present paper, the sequential time will be measured running DAMNED on a single machine (*i.e.* a single simulation task). Of course the machine is part of the cluster.

$$S_P = \frac{best\ T_1}{T_P} \tag{1}$$

However, measuring $T_1$ is not easy for large SNN since a single processor does not have enough memory. Therefore, we have been running a 10 000 and 100 000 neurons network but only small networks runs within the 1 GB of a single computer. As a consequence a theoretical sequential time has been extrapolated for large networks, according to a function $f(n) = \alpha\ n^2 + \beta\ n + \gamma$ where $n$ is the size of the network. Such an estimation assumes an infinite memory for the sequential machine. The parameters $\alpha$, $\beta$ and $\gamma$ have been fitted running DAMNED on a single machine for 1 000, 2 000, 3 000, 4 000, 7 000, and 10 000 neurons at 1 Hz and 10 Hz. The 4 000 neurons network is the largest before the OS starts to swap which leads to unusable simulation durations. For 7 000 and 10 000 neurons, the physical memory of the machine has been doubled with an identical memory module taken from an other machine of the cluster, reaching the maximum amount of memory possible for our machines regarding the available memory modules (only 2 memory slots, populated with 1 GB modules). The function $f$ fits perfectly the data with $\alpha = 7.01\ 10^{-6}$, $\beta = 0.118$ and $\gamma = -116$ at 10 Hz. At 1 Hz we found $\alpha = 1.86\ 10^{-6}$, $\beta = 8.42\ 10^{-3}$ and $\gamma = -6.33$. The same methodology has been used to estimate the sequential time of the creation of the SNN, which does not depend of the frequency, giving $\alpha = 1.09\ 10^{-5}$, $\beta = -0.0346$ and $\gamma = 34$. Creation speedups are presented in the conclusion.

Figure 4 presents speedups as a function of the number of simulation tasks. No less than 15 tasks are launched for the 10 000 neurons network since enough

machines are needed for the OS not to swap (see both dashed lines on figure 4). Making larger neural networks runnable is the main successful goal of DAMNED. Furthermore, results show that simulation times are significantly decreased, even with the low cost 100 Mb/s network used. The two 10 000 neurons simulations (dashed lines) are performed at 1 Hz and 10 Hz. Both frequencies are biologically plausible average values, however some cells can spike at a higher rate. Results show that a higher activity implies lower speedups. Indeed it is more difficult for computations to hide the increased number and size of messages. Thus dealing with high average frequencies would take advantage of a better network. Finally, the 100 000 neurons network (see plain line), involving at least 27 simulations tasks, validates the scalability of DAMNED for effective simulations.

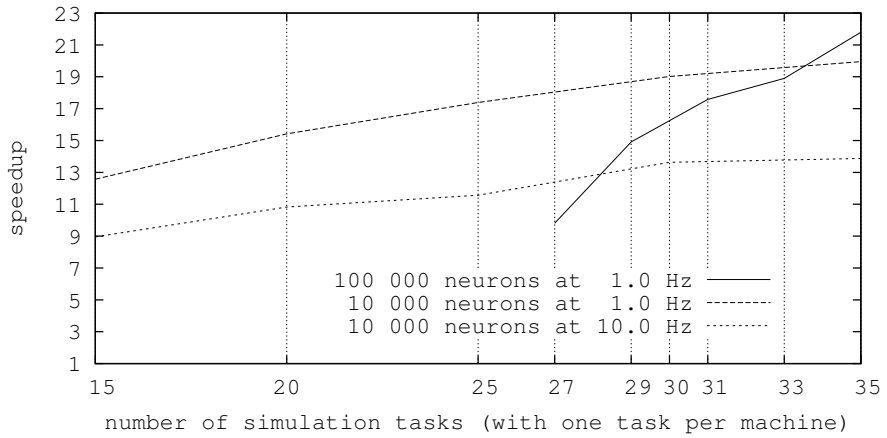


**Fig. 4.** Speedups as a function of the number of simulation tasks, with average spiking rates of 1 Hz and 10 Hz, with biologically plausible connectivity of respectively 0.1 % and 0.01 % of the network size for a 10 000 and a 100 000 neurons networks

## 7 Conclusion and Future Work

This paper has presented the DAMNED architecture and some original contributions to the field of distributed event-driven simulations, namely an optimised delayed event queue and efficient distributed virtual time handling algorithms. DAMNED is well adapted for biologically inspired protocols and can output valuable simulation data. Significant speedups have been achieved and prove that even using a non dedicated cluster of simple computers, the DAMNED simulator is able to handle large SNN simulation. Such results tend to validate the scalability of DAMNED. Lower speedups occur when the computational load is not high enough to overlap communications, thus complex neuron models would definitively lead to high speedups. Furthermore, results show that even when DAMNED is under-loaded, increasing of the number of processor does not slow the simulation which can in all cases reach a normal end. More evaluations will be done on dedicated parallel computers to simulate larger SNN.

Figure 5 presents the speedups of the creation of the SNN and shows that this step already takes advantage of the cluster. However the creation time remains high for large neural networks, about 2 000 s for a 100 000 neurons network using 35 machines while simulating 1 s of biological time takes about 150 s. We are currently working on this issue and significant improvements are anticipated.

Regarding usability, DAMNED is end user oriented and the creation of a SNN has been presented. A web based interface which enables to launch simulations with a simple navigator is already developed and will be improved. This interface shows available machines on the cluster, creates MPI configuration files, runs DAMNED and collects results. For now, the main remaining difficulty for an end-user is the modification of neuron models and implementation of new models, witch will be addressed by defining new neuron models from the web interface.

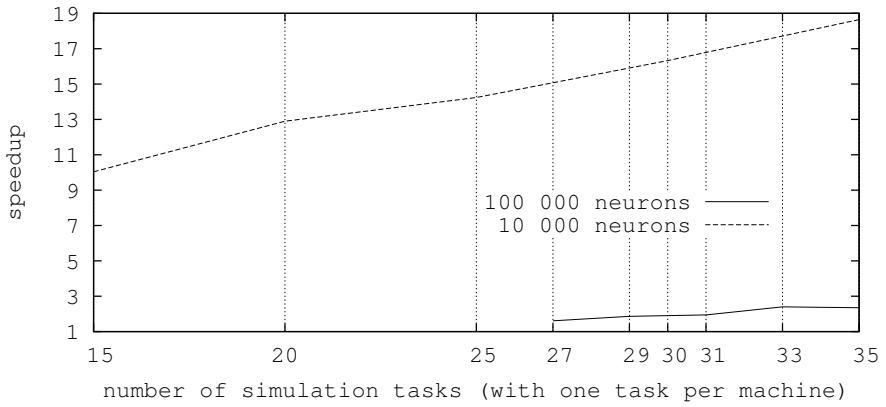


**Fig. 5.** Speedups of the creation of the SNN for a 10 000 and a 100 000 neurons network

## References

[1] Bohte, S.M.: The evidence for neural information processing with precise spike-times: A survey. Natural Computing 3(4), 195–206 (2004)
[2] Chandy, K.M., Misra, J.: Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering, SE-5(5), 440–452 (1979)
[3] Ferscha, A.: Parallel and distributed simulation of discrete event systems. In: Parallel and Distributed Computing Handbook, pp. 1003–1041. McGraw-Hill, New York (1996)
[4] Flynn, M.J., Rudd, K.W.: Parallel architectures. ACM Computation Surveys 28(1), 67–70 (1996)
[5] Message Passing Interface Forum. MPI: A message-passing iterface standard. Technical Report UT-CS-94-230, University of Tennessee (1994)
[6] Gerstner, W., Kistler, W.M.: Spiking Neuron Models: An Introduction. Cambridge University Press, New York (2002)
[7] Izhikevich, E.M.: Simple model of spiking neurons. IEEE Transactions on Neural Networks 14(6), 1569–1572 (2003)

[8] Knight, B.W.: Dynamics of encoding in a population of neurons. Journal of General Physiology 59, 734–766 (1972)
[9] Lobb, C.J., Chao, Z.C., Fujimoto, R.M., Potter, S.M.: Parallel event-driven neural network simulations using the hodgkin-huxley model. In: Proceedings of the Workshop on Principles of Advanced and Distributed Simulations. PADS 2005, June 2005, pp. 16–25 (2005)
[10] Makino, T.: A discrete-event neural network simulator for general neuron models. Neural Computing and Applications 11(3-4), 210–223 (2003)
[11] Marin, M.: Comparative analysis of a parallel discrete-event simulator. In: SCCC, pp. 172–177 (2000)
[12] Mattia, M., Giudice, P.D.: Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. Neural Computation 12, 2305–2329 (2000)
[13] Morrison, A., Mehring, C., Geisel, T., Aertsen, A., Diesmann, M.: Advancing the boundaries of high-connectivity network simulation with distributed computing. Neural Computation 17, 1776–1801 (2005)
[14] Rudolph, M., Destexhe, A.: Analytical integrate-and-fire neuron models with conductance-based dynamics for event-driven simulation strategies. Neural Computation 18(9), 2146–2210 (2006)
[15] Shelley, M.J., Tao, L.: Efficient and accurate time-stepping schemes for integrate-and-fire neuronal network. Journal of Computational Neuroscience 11(2), 111–119 (2001)
[16] Swadlow, H.A.: Efferent neurons and suspected interneurons in binocular visual cortex of the awake rabbit: Receptive fileds and binocular properties. Journal of Neurophysiology 59(4), 1162–1187 (1988)
[17] Watts, L.: Event-driven simulation of networks of spiking neurons. In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing System, vol. 6, pp. 927–934. MIT Press, Cambridge (1994)