Parallel and pseudorandom discrete event system specification vs. networks of spiking neurons: Formalization and preliminary implementation results

Alexandre Muzy
CNRS I3S UMR 7271
06903 Sophia-Antipolis Cedex, France.
Email: alexandre.muzy@cnrs.fr

Matthieu Lerasle, Franck Grammont
Univ. Nice Sophia Antipolis
CNRS LJAD UMR 7351
06100 Nice, France.

Van Toan Dao, David RC Hill
ISIMA/LIMOS UMR CNRS 6158
Blaise Pascal University
BP. 10125, 63173 AUBIERE Cedex, France.

***Abstract*—Usual Parallel Discrete Event System Specification (P-DEVS) allows specifying systems from modeling to simulation. However, the framework does not incorporate parallel and stochastic simulations. This work intends to extend P-DEVS to parallel simulations and pseudorandom number generators in the context of a spiking neural network. The discrete event specification presented here makes explicit and centralized the parallel computation of events as well as their routing, making further implementations more easy. It is then expected to dispose of a well defined mathematical and computational framework to deal with networks of spiking neurons.**

***Index Terms*—Spiking neuron networks, discrete event system specification, pseudorandomness, parallel simulation, multithreading.**

I. Introduction

Discrete events allow faithfully implementing spike exchanges between biological neurons. Discrete event spiking neurons have been widely implemented in several software environments [3]. However, as far as we know, there is no attempt to embed these works into a common mathematical framework that would allow further theoretical and practical developments between biology and computer science. To achieve this goal the Parallel Discrete Event System Specification (P-DEVS) [2], [6] is used here. This framework provides well defined structures for the formal and computational specifications of a general dynamic system structure. However, using P-DEVS directly in the context of networks of spiking neurons requires first the development of:

- New (general) formal structures to capture explicitly and unambiguously the parallel and stochastic aspects of spiking neural networks.
- New simple and abstract algorithms for the parallelization of discrete event computations and exchanges.

This study aims at answering the following questions: What are the main computational loops to be parallelized in a DEVS simulator? How to manage rigorously the stochastic aspects of these parallel simulations? What are the corresponding mathematical structures? How these elements can be used to simulate networks of spiking neurons?

More precisely, we propose here:

- A formalization of networks as parallel discrete event system specifications using pseudorandom generators for the generation of stochastic trajectories and network structures,
- A simple parallelization technique of events in P-DEVS simulators,
- An application of all these concepts to random networks of spiking neurons.

The manuscript is organized as follows. In section 2, the formal model, simulator and executor algorithms are presented. In section 3, stochastic, parallel and pseudorandom generator structures are defined. In section 4, a spiking neural network model and its discrete event system specification are presented. In section 5, simulation process and results are presented and discussed. Finally, conclusion and perspectives are provided.

II. Modeling and simulation framework

In this section the main elements for modeling and simulation are detailed in a parallel processing scope.

A. Architecture

Figure 1 presents an architecture for modeling, simulation and execution. In part a), the use of a Middleware is conceived as a communication interface between Software and Hardware components. In part b), an extension of the usual model/simulator separation [2] to hardware interfacing is presented. The Model specifies the elements of a dynamic system for digital computers. The Simulator generates the behavior of the Model. The added Executor runs the simulation computations on each available Processor (or core).

Benefits from usual middleware (in software engineering) and model/simulator separation (in modeling and simulation) are well known in terms of model reusability. These concepts are extended here to include hardware management.

More precisely, in part b):

- Between the Model and the Simulator: There is a connection between the elements of the Model and the

corresponding elements of the Simulator, possibly using different simulators for the same model and *vice versa*. In the reverse direction, the behavior generation of the Model is achieved by the Simulator;

- Between the Simulator and the Executor: There is a submission of tasks (possibly using different executors for the same simulator and *vice versa*). In the reverse direction, a selection of the Simulator elements is achieved by the Executor;
- Between the Executor and the Processor: There is a supervision of the Processor (physical or logical) attributing each task to an available Processor. In the reverse direction, the result of each operation is sent back from the Processor to the Executor.

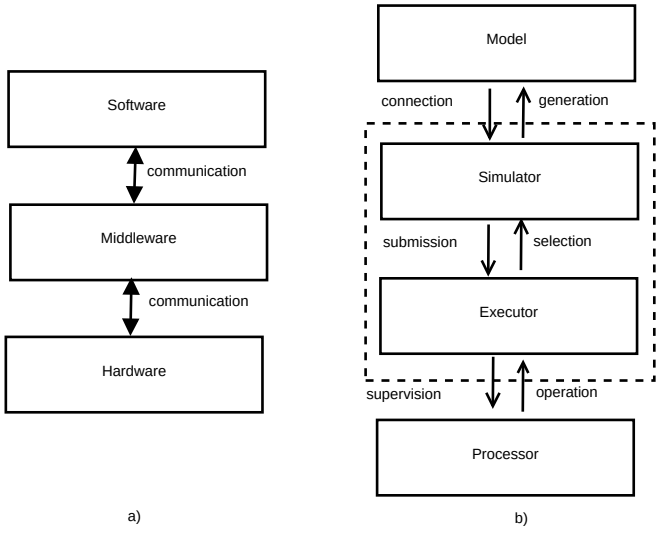


Figure 1. Architecture for modeling, simulation and execution: a) Software engineering, b) Extension of the usual model/simulator separation to hardware interfacing.

In the next subsections each element and link of the architecture for modeling, simulation and execution is detailed.

B. Model

Definition 1. A *basic Parallel Discrete Event System Specification (P-DEVS)* is a mathematical structure

$$\textit{P-DEVS} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

Where, X is the *set of input events*, Y is the *set of output events*, S is the *set of partial states*, $\delta_{ext} : Q \times X^b \to S$ is the *external transition function* with $Q = \{(s, e) \,|\, s \in S,\, 0 \leq e \leq ta(s)\}$ the set of total states with e the *elapsed time* since the last transition, $\delta_{int} : S \to S$ is the *internal transition function*, $\delta_{con} : S \times X^b \to S$ is the *confluent transition function, where X^b is a bag of input events,* $\lambda : S \to Y^b$ is the *output function*, *where Y^b is a bag of output events,* and $ta : S \to \mathbb{R}^{0,+}_{\infty}$ is the *time advance function.*

The modeler controls/decides the behavior in case of event collisions, when the basic system, at the same simulation time, is concerned by both internal and external events. To do so, the modeler defines the *confluent transition function* δ_{con}.

Example 2. Simple P-DEVS dynamics

In Figure 2, it is considered that at time t_2, there is no collision between external event x_0 and the internal event scheduled at time $ta(s_1) = t'_2$, with $t'_2 > t_2$, thus leading to an external transition function $\delta_{ext}(s_1, e_1, x_0) = s_2$. At time $ta(s_3) = t_4$ where there is a collision between external event x_1, occurring at time t_4, and the internal event scheduled at the same time thus leading to a confluent transition function: $\delta_{con}(s_3, x_1) = s_4$.

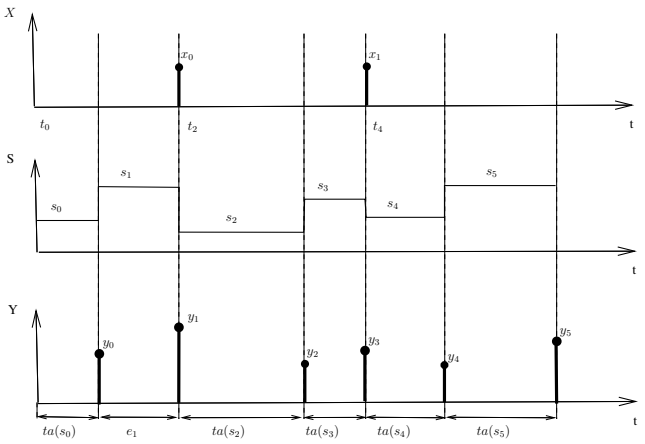


Figure 2. Simple *P-DEVS* trajectories.

Definition 3. A *P-DEVS network* is a mathematical structure

$$N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$$

Where, X is the *set of input events*, Y is the *set of output events*, D is the set of *component names*, for each $d \in D$, M_d is a *basic or network model*, I_d is the set of *influencers* of d such that $I_d \subseteq D$, $d \notin I_d$ and, for each $i \in I_d$, $Z_{i,d}$ is the $i{-}to{-}d$ *output translation*, defined for: (i) *external input couplings*: $Z_{self,d} : X_{self} \to X_d$, with $self$ the self network name, (ii) *internal couplings*: $Z_{i,j} : Y_i \to X_j$, and (iii) *external output couplings*: $Z_{d,self} : Y_d \to Y_{self}$.

C. Simulator

Algorithm 1 describes the main simulation loop of a P-DEVS model. The hierarchical structure (i.e., the composition of nested network models finally composed of basic models) is made implicit here by manipulating the set of component names (referring to all the components present in the hierarchy). This is made possible because a P-DEVS network is closed under coupling, i.e., the behavior of a P-DEVS network is equivalent to the behavior of a P-DEVS basic model resultant. In the main-loop algorithm, as in a usual discrete event simulation, simulation time advance is driven by the (last and next) times of occurrence of events. Three component sets allow focusing concisely and efficiently on active components at each time step of the simulation: (i) the *imminent set* $IMM(s)$ (the set of components that

achieve both an output computation and an internal function transition), (ii) the *sender set* $SEN(s)$ (the set of components that *actually* send output events to the components they are connected with), and (iii) the *receiver set* $REC(s)$ (the set of components that receive output events). The Executor is in charge of the execution of: initialization, outputs, routing, and confluent, external and internal transitions; as well as the determination of the set of the next times of event occurrences and the imminent set.

Algorithm 1 Main simulation loop of Root Coordinator.

Variables:
t_l: Global time of last event
t_n: Global time of next event
$s = (\dots, (s_d, e_d), \dots)$: Global state
$TNEXT(s) = \{t_{n,d} \,|\, d \in D\}$: set of times of next events
$IMM(s) = \{d \in D \,|\, t_{n,d} = t_n\}$: set of imminents for next output/internal transition
$SEN(s) = \{d \in D \,|\, \lambda_d(s_d) \neq \emptyset\}$: set of senders
$REC(s) = \{d \in D \,|\, i \in I_d \wedge i \in IMM(s) \wedge x_d^b \neq \emptyset \wedge Z_{i,d}(x_d^b) \neq \emptyset\}$: set of receivers
n_{LWP}: number of lightweight processes

Begin
$t_l \leftarrow 0$
$Executor.$*self-init*(n_{LWP})
$Executor.$*initialize*(D)
$TNEXT(s) \leftarrow Executor.$*get-TN*$(D)$
$t_n \leftarrow min(TNEXT(s))$
while $t_n < t_{end}$ **do**
$IMM(s) \leftarrow Executor.$*getImminents*$(D, t_n)$
$SEN(s) \leftarrow Executor.$*computeOutputs*$(IMM(s), t_n)$
$REC(s) \leftarrow Executor.$*route*$(SEN(s))$
$ACTIVE(s) \leftarrow IMM(s) \cup REC(s)$
$Executor.$*computeTransitions*$(ACTIVE(s), t_n)$
$t_l \leftarrow t_n$
$TNEXT(s) \leftarrow Executor.$*get-TN*$(D)$
$t_n \leftarrow min(TNEXT(s))$
end while
End

D. Executor

The executor acts as an interface between the simulation and the hardware execution. As described in Algorithm 2, the executor implements the execution of $TASKS$ (i.e., functions over components) attributing each of them to an available lightweight process $lwp \in LWP$ (here a thread). If the simulator deals with simulation time *t*, models and simulator nodes, the executor deals with *execution time* t_{exec}, lightweight processes and *logical cores* (or processors). At the initialization of the executor if the *number of lightweight processes* n_{LWP} is greater than the *number of logical cores,* n_{LC}, of the machine, then $n_{LWP} = n_{LC}$[1]. The set $TASKS$ implements functions/procedures init, get-TN, compute-outputs, route and compute-transitions for each component $d \in D$. The execution can be sequential ($n_{LWP} = 1$) or parallel ($n_{LC} \geq n_{LWP} > 1$).

[1]Otherwise the parallelization will be inefficient. Also it is expected then that the operating system assigns available cores to available lightweight processes.

III. Stochastic discrete event system specification

The formal structures reflecting the discreteness of the computations achieved by digital computers are presented here. First, a general generator definition based on sequential machines is presented. Based on this definition, a structure for pseudorandom number generators is proposed and linked to the definition of pseudorandom variables. Using a pseudorandom number generator, a pseudorandom variate generator is used for computing the realizations of pseudorandom variables. A pseudorandom and parallel event execution is specified in P-DEVS. At structural level, large numbers of connections and components in a network are captured using a pseudorandom graph definition. The latter is finally compared to the P-DEVS network structure definition.

A. Systems, pseudorandom variables and pseudorandom number generators

Definition 4. A *generator* is an autonomous sequential machine $G = (S, s_0, \gamma)$, where S is the *set of states*, s_0 is the *initial state* and $\gamma : S \to S$ is the *state generation function*.

Definition 5. A *pseudorandom number generator (RNG)* (cf. [6], p.132, whose definition is extended here) is defined as $RNG = (S_P, s_{P_0}, \gamma_P)$, with $S_P = R$ the *generator state set* with $R \subset \mathbb{R}_{[0,1]}$ the *finite set of pseudorandom numbers* (with each pseudorandom number a realization of a uniformly distributed random variable, i.e., $r \sim \mathcal{U}(0,1)$), $\gamma_P : R \to R$ the *pseudorandom number generation* map, and $s_{P_0} = r_0$ the *initial status* (or *seed* for old generators). A *stream (i.e., a sequence) of independent and identically distributed (i.i.d.) pseudorandom numbers of length period l*, noted $(r_i)_{i=0}^{l-1} = r_0, r_1, \dots, r_{l-1}$, for $i = 0,\ 1, \dots,\ l-1$, is *defined by* $\gamma_P(r_i) = r_{i+1}$ and with $\gamma_P(r_{l+i}) = r_i$.

Definition 6. *A pseudorandom variate generator (RVG)* is defined as $RVG = (RNG, S_V, s_{V_0}, \gamma_V)$, with $S_V = V$ the *generator variate set*, $V \subset \mathbb{R}$ the *finite set of pseudorandom variates* (with each random variate $v \in V$ being a realization of a random variable with inverse non-uniform cumulative function distribution γ_V), $\gamma_V : R \to V$ the *pseudorandom variate generation* map, and s_{V_0} the *initial pseudorandom variate.* A *stream of pseudorandom variates follows exactly the sequence of the pseudorandom numbers generated by RNG and is of equal length l*, i.e., for $(r_i)_{i=0}^{l-1} = r_0, r_1, \dots, r_{l-1}$, there exists $(v_i)_{i=0}^{l-1} = v_0, v_1, \dots, v_{l-1}$.

Definition 7. *A pseudorandom variable* consists of the *map* $\gamma_V : R \to V$ of a *pseudorandom variate generator* $RVG = (RNG, S_V, s_{V_0}, \gamma_V)$, where $R \subset \mathbb{R}_{[0,1]}$ is a *finite set of uniformly distributed pseudorandom numbers.* Every time a *random variate* $v_i \in V$ *of* the pseudorandom variable $\gamma_V(r_i)$ is obtained, the next pseudorandom number is generated through $r_{i+1} = \gamma_P(r_i)$.

Algorithm 2 Variables, procedures and functions of Executor.

Variables:

$LWP = \{lwp \,|\, n_{LWP} \leq n_{LC}\}$: set of lightweight processes

n_{LWP}: number of lightweight processes

n_{LC}: number of logical cores

$TASKS = \{f_d \,|\, d \in D\}$: set of tasks with f_d a function to execute over $d \in D$

t_{exec}^{max}: maximum execution time of a process

Begin

procedure SELF-INIT(n_{LWP})
  $n_{LC} \leftarrow getNbOfLogicalCores()$
  if $n_{LWP} > n_{LC}$ **then**
    $n_{LWP} \leftarrow n_{LC}$
  end if
end procedure
function RUN($TASKS$)
  In parallel $\forall task \in TASKS$ **do**
    run $task$ on available $lwp \in LWP$
    add possibly result **to** ResSet
  end In parallel
  lock $TASKS$
  wait t_{exec}^{max} for each $lwp \in LWP$ to terminate
  return ResSet
end function
procedure INIT(D)
  set $TASKS = \{(d, init(0)) \,|\, d \in D\}$
  $run(TASKS)$
end procedure
function GET-TN(D)
  $TASKS = \{(d, getTn()) \,|\, d \in D\}$
  $TNEXT(s) \leftarrow run(TASKS)$
  return $TNEXT(s)$
end function
function COMPUTE-OUTPUTS($IMM(s), t_n$)
  set $TASKS = \{(imminent, computeOutput(t_n)) \,|\, imminent \in IMM(s)\}$
  $SEN(s) \leftarrow run(TASKS)$
  return $SEN(s)$
end function
function ROUTE($SEN(s)$)
  $TASKS = \{(sender, route()) \,|\, sender \in SEN(s)\}$
  $REC(s) \leftarrow run(TASKS)$
  return $REC(s)$
end function
procedure COMPUTE-TRANSITIONS($ACTIVE(s), t_n$)
  $TASKS = \{(active, computeDelta(t_n)) \,|\, active \in ACTIVE(s)\}$
  $run(TASKS)$
end procedure

End

Example 8. For a pseudorandom variable following an exponential law $\gamma_V \sim Exp(\lambda)$, each realization (pseudorandom variate) is obtained by $\gamma_V(r) = \frac{-ln(1-r)}{\lambda} = v$ (i.e., inverting the cumulative distribution function of the exponential law).

B. Pseudorandom Parallel Discrete Event System Specification

As previously defined, randomness is simulated at the computer level using a pseudorandom number generator modeled as a deterministic sequential machine. Corresponding pseudorandom variables are maps taking the generated pseudorandom numbers in argument and generating corresponding pseudorandom variates. At formal P-DEVS level, the set of pseudorandom variates V can be embedded as part of the partial state.

Definition 9. A *basic Pseudorandom Parallel Discrete Event System Specification* (*PP-DEVS*) is a structure

$$\textit{PP-DEVS} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

Where, X and Y defined previously, $S \supseteq V$ is the *set of sets of global pseudorandom variates* $V = \Pi_{i=1}^{n} V_i$ with *n* the number of pseudorandom variables. Each set V_i contains the pseudorandom variates of a stream $(v_i)_{i=0}^{l-1} = v_0, v_1, \ldots, v_{l-1}$ generated by a corresponding pseudorandom variate generator $RVG_i = (RNG_i, S_{V_i}, s_{V_{i,0}}, \gamma_{V_i})$ (cf. Definition 6), thus defining a pseudorandom variable $\gamma_{V_i} : R_i \rightarrow V_i$. At each transition function execution the next state is computed *deterministically* based on a global pseudorandom variate $v \in V$ and a partial state $s \in S$, i.e., $\delta_{int}(s, v) = s'$, $\delta_{ext}(q, x, v) = s'$, and $\delta_{con}(s, x, v) = s'$[2].

The use of pseudorandom numbers in deterministic sequential DEVS models has been discussed in the context of probability spaces [4]. This work pinpointed cases where the previous definition may show inconsistencies as well as convergence issues (when elements are not measurable or corresponding sets infinite). However, our goal here is not to redefine a new formalism at continuous system specification level but rather to specify the deterministic foundations of the stochastic simulations achieved at computer level and how this can be modeled in P-DEVS as a first step. This does not prevent to achieve further mathematical extensions, as done in [4].

C. Random graph-based network

Definition 10. A *pseudorandom directed graph generator (RGG)* is a structure $RGG = (\mathcal{G}^{n,p}, S_G, s_{G_0}, \gamma_G)$, where $S_G = V_{coupling} = \mathbb{B}$ is the *set of coupling pseudorandom variates* obtained by sampling corresponding (Bernouilli) *coupling pseudorandom variable* $\gamma_{coupling} \sim \mathcal{B}(p)$[3], $s_{G_0} = v_{coupling,0}$

[2]The same reasoning can be done based on each pseudorandom number $r_i \in R_i$, such that the set of sets of (global) pseudorandom numbers is $R = \Pi_{i=1}^{n} R_i$, with *n* the number of pseudorandom numbers. Then, at each transition function execution the next state of *P-DEVS*$_R$ is computed based on each global pseudorandom number $r \in R$, i.e., $\delta_{int,R}(s, r) = s'$, $\delta_{ext}(q, x, r) = s'$, and $\delta_{con}(s, x, r) = s'$. Notice here that each pseudorandom number defines a probability of external, internal and confluent transition.

[3]Pseudorandom variates in S_G are generated by a pseudorandom variate generator $RVG_{coupling} = (RNG_{coupling}, V_{coupling}, v_{coupling,0}, \gamma_{coupling})$.

is the *initial coupling pseudorandom variate*, $\mathcal{G}^{n,p}$ is the *set of all pseudorandom directed graphs* such that $\mathcal{G}^{n,p} = \mathcal{G}\{n, P(arrow) = p\}$, with *n* the *number of vertices* and $p \in \mathbb{R}_{[0,1]}$ the *probability of choosing an arrow*. Each directed graph $G(U, A) \in \mathcal{G}^{n,p}$ is described by $U = \{1, 2, \ldots, n\}$ the *set of vertices* and A (a set of *ordered pairs*) the *set of arrows*. Last map $\gamma_G : \mathcal{G}^{n,p} \times S_G \to \mathcal{G}^{n,p}$ is the *directed graph generation map* using the coupling pseudorandom variates $V_{coupling}$ to construct a graph $G(U, A) \in \mathcal{G}^{n,p}$.

Once the graph structure has been generated, the graph can be transformed into a network where to each node corresponds a P-DEVS component and to each arrow a coupling.

Definition 11. A *Graph-to-P-DEVS Network Transformer* (GNT) is a structure $GNT = (G, N, \{m_{i,j}\})$, where G is a *directed graph*, N is a *P-DEVS network*, $m_{i,j}$ is a *one-to-one map* (from the elements of G to the elements of N) defined for: (i) *vertices-to-components* $m_{u,c} : U \to D$, (ii) *arrows-to-couplings* $m_{a,c} : U \times U \to D \times D$ with $D \times D = \{(a, Z_{a,b}(a)) \,|\, a \in I_b\}$ the influencer-to-influencee pairs, and (iii) *arrows-to-influencers* $m_{a,i} : U \times U \to \{I_i\}$ with $m_{a,i}(u, u') \in I_{u'}$ the selection of the influencer of u'.

IV. Spiking neural network model

Mathematical modeling of a random spiking neural network is presented here. The model is specified after using the main mathematical structures presented in previous sections.

A. Biological neuron

Figure 3 depicts a single biological neuron. Most commonly, inputs from other neurons are received on *dendrites*, at the level of *synapses*. The circulation of neuronal activity (electric potentials) is due to the exchange through the neuron *membrane* of different kinds of ions. Dendrites integrate locally the variations of electric potentials, either excitatory or inhibitory, and transmit them to the *cell body*. There, the genetic material is located into the *nucleus*. A new pulse of activity (an *action potential*) is generated if the local electric potential reaches a certain threshold at the level of the *axon* hillock, the small zone between the cell body and the very beginning of the axon. If emitted, action potentials continue their way through the axon in order to be transmitted to other neurons. Action potentials, once emitted, are "all or nothing" phenomena: $0, 1$. The propagation speed of action potentials can be increased by the presence of a *myelin* sheath, produced by *Schwann cells*. This insulating sheath is not continuous along the axon. There is no myelin at the level of the *nodes of Ranvier*, where ionic exchanges can still occur. When action potentials reach the tip of the axon, they spread over all *terminals* with the same amplitude, up to synapses. The neuron can then communicate with other following neurons. Notice that a focus on electrical signals (without dealing with chemical signals) is achieved here.

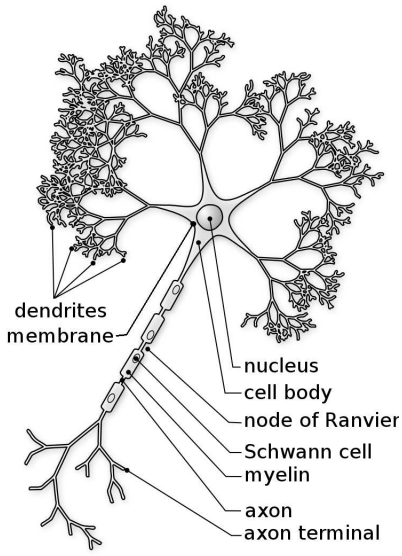


Figure 3. Sketch of a neuron (adapted from http://fr.wikipedia.org/wiki/Neurone).

B. Model

Definition 12. The graphs structure (cf. Figure 4)

Let I, B, O be 3 finite sets with respective cardinality n, M and N. It is always assumed that $M \geq N$. Let $(p_i)_{i \geq 0}$ denote real numbers in $[0, 1]$. For any $(i, j) \in B^2$, assume that there exists an arrow $i \to j$ with probability p_0, for any $i \in I$ and $j \in B$, assume that there exists an arrow $i \to j$ with probability p_1 and for any $i \in B$ and $j \in O$, assume that there exists an arrow $i \to j$ with probability p_2.

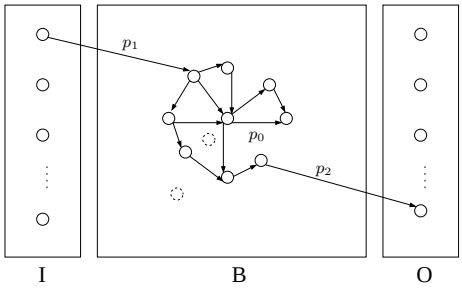


Figure 4. Structure of the neuron model.

Definition 13. The dynamics

Assume that the activities $(X_t(i))_{i \in I, t \in \mathbb{N}}$ of the sites in I and time t are i.i.d. $\mathcal{B}(p_3)$. Let a be a positive real number. For all $(i, j, t) \in (B \cup O)^2 \times \mathbb{N}$, we choose i.i.d. thresholds $\tau_i \sim \mathcal{N}(m, \mathcal{S}^2)$, i.i.d. $w_{i,j} = 1$ with probability 0.8 and $-a$ with probability 0.2. Then, the *membrane potential* $P_i(t)$ of a neuron i, initially null is updated thanks to the following rule

$$P_i(t) = (rA_i(t-1) + \sum_{i \sim j} w_{i,j} A_j(t-1))(1 - A_i(t-1))$$

Where, $r \in (0, 1)$ is the *activity remaining from time* $t - 1$, $\sum_{i \sim j} w_{i,j} A_j(t-1)$ is the *activity received from other neurons*

at time $t-1$, $(1-A_i(t-1))$ reflects a *refractory period* of 1 (if the neuron fired at time $t-1$ it cannot fire at time t), and the *activity of a neuron* i is provided by

$$A_i(t) = \begin{cases} 1 & if\ P_i(t-1) \geq \tau_i \\ 0 & otherwise \end{cases}$$

C. Specification in PP-DEVS

Each neuron $i \in I$ is specified as a PP-DEVS reduced to internal transitions as

$$M_i = (Y_i, S_i, \delta_{int,i}, \lambda_i, ta_i)$$

Where, $Y_i = \{\emptyset, 1\}$, with null event $\emptyset$ (resp. 1) if the neuron is non-firing (resp. firing), $S_i = V_{firing} = \mathbb{B}$ with V_{firing} the set of firing pseudorandom variates, *internal transition function* $\delta_{int,i}(s, v_{firing})$ samples the pseudorandom variable $\gamma_{firing} \sim \mathcal{B}(p_3)$ indicating the activity of the neuron depending on probability p_3, output function $\lambda_i(v_{firing})$ sends an unitary event if the neuron is active and *time advance function* $ta_i(s) = 1$ ensures the discrete time sampling of γ_{firing}.

Neurons in B and O are P-DEVS models specified as

$$M_j = (X_j, Y_j, S_j, \delta_{ext,j}, \delta_{int,j}, \delta_{con,j}, \lambda_j, ta_j)$$

Where, $X_j = \{\emptyset, 1\}^n = \{\emptyset, 1\} \times \ldots \times \{\emptyset, 1\}$ (with n the *number of inputs*), $Y_j = \{\emptyset, 1\}$, $S_j = \{\{w_k\}, c, a, p, phase = \{\mathsf{firing}, \mathsf{active}, \mathsf{inactive}\}\}$ with w_j the *weight of corresponding input* k, c (resp. c') the *sum of received inputs* at a time step t (resp. at a time step $t+1$), a (resp. a') the *activity of the neuron* at a time step t (resp. at time step $t+1$), p (resp. p') the *membrane potential* of the neuron at a time step t (resp. at time step $t+1$), *external transition function* $\delta_{ext}(q, x)$ collects the inputs received at time t, computes the next phase and the next membrane potential p' and activity a', and after call for a next internal transition at time $t+1$, *internal transition function* $\delta_{int}(s)$ updates $p \leftarrow p'$ and $a \leftarrow a'$ and reset inputs ($c \leftarrow 0$), if the neuron is in phase active or firing and receives an input the confluent transition function is called as $\delta_{con}(s, x) = \delta_{ext}(\delta_{int}(s), 0, x)$, i.e., first update variables and after collect inputs, and finally *time advance function* $ta_j(s) = 1$ if the neuron is in phase active or firing and $ta_j(s) = \infty$ if the neuron is in phase inactive.

The graph structure of neurons in B is generated by a *pseudorandom directed graph generator* (RGG_N) with p_0 the *probability of choosing an* arrow.

V. Simulation process and results

Model generation and simulation process are introduced first here. After, the speed-up results are presented and discussed.

A. Environment infrastructure and graphical outputs

The steps and the elements of the process of generation and simulation of the model consist of the following sequence: (i) Initialization of all models, (ii) Graph generation using a model RGG, (iii) Graph-to-network transformation (GNT), which generates a PP-DEVS network from the graph, and (iv) Simulation.

Notice that as defined previously, each object uses one RNG for each pseudorandom variable. This ensures: (i) the statistical independence between pseudorandom variables, and (ii) the reproducibility of pseudorandom simulations [5].

Figure 5 depicts a snapshot of the graph corresponding to neurons of set *B*. Notice how dense is the graph connection making it difficult to differentiate edges.

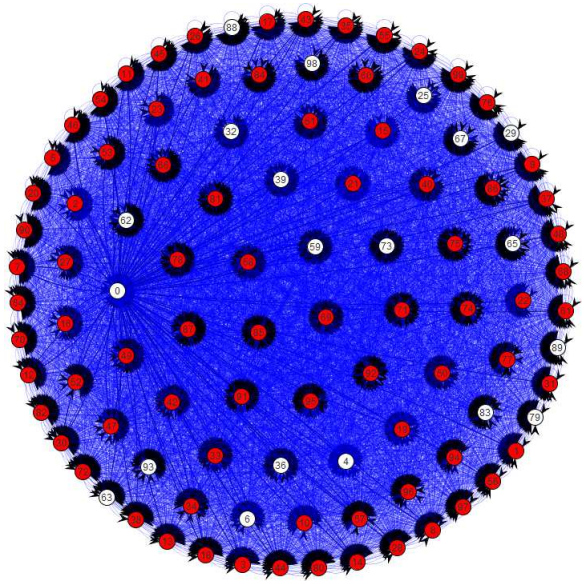


Figure 5. Graph snapshot of *B* set.

Simulations have been performed on a Symmetric Multiprocessing (SMP) machine with *80* physical cores and *160* logical cores, 8 processors Intel(R) Xeon(R) CPU *E7-8870@2.40GHz*[4], and *1Tb* RAM. Figure 6 presents the firing of neurons for neurons of each set *I*, *O*, and *B*.

B. Speed-up results

In [7], an interesting perspective is drawn concerning the usage of clusters with low latency communication capabilities. Our idea here is to assume (even at abstract simulator level) that all the computations are centralized on a single computer, a Symmetric Multiprocessing (SMP) machine[5]. The latter allows sharing memory and minimizing the latency of communications. Besides, centralizing all the computations facilitates the control of their executions and their synchronization at each time step. Different sizes of neural networks are simulated here for different numbers of threads.

Input parameters are set to values: $p_0 = p_1 = p_2 = 0.9$, $p_3 = 0.5$, $a = r = 1$, each threshold $\tau_i \sim \mathcal{N}(m, \mathcal{S}^2)$,

[4]stepping: 2, cpu: 1064 MHz, cache size: 30720 KB.

[5]Simulations have been performed on a Symmetric Multiprocessing (SMP) machine with *80* physical cores and *160* logical cores, 8 processors Intel(R) Xeon(R) CPU *E7-8870@2.40GHz* (stepping: 2, cpu: 1064 MHz, cache size: 30720 KB.), and *1Tb* RAM. Each Java class main has been executed in command line using the exec-maven-plugin-1.2.1. Execution times correspond to the total (processor) time information provided by Maven. Finally, when running the simulations, the machine was possibly executing other simulations (launched by other users).

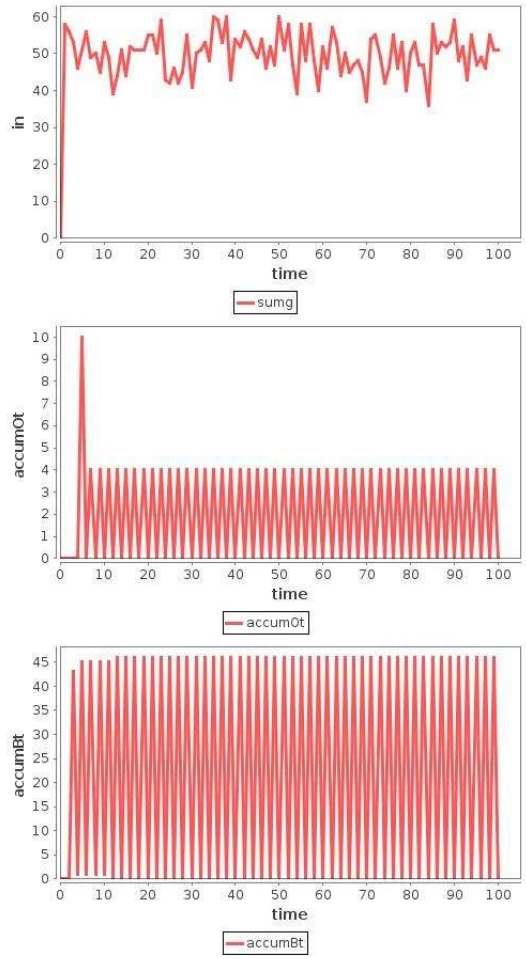


Figure 6. Firing outputs in sets *I*, *O*, and *B*.

with $m = 250$ and $\mathcal{S} = 1$. The whole simulation has been implemented in Java programming language.

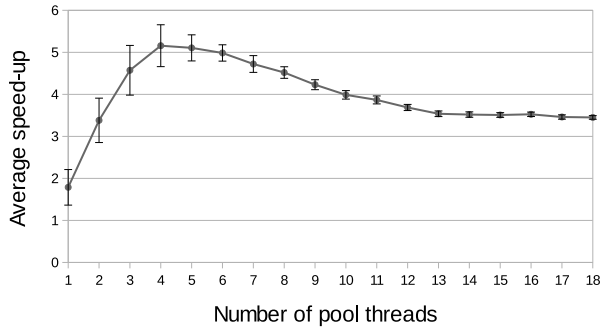
The *sequential execution time of methods* $t_{methods}$ has been considered as the sum of the execution times for methods: *initialization* (t_{init}), *output* (t_{out}), *routing* (t_{rout}), and *transitions* (t_{trans}) (cf. Algorithm 1), for different sizes of networks. Considering t_{total} as the *total parallelizable execution time*, and t_{seq} as the *sequential execution time that cannot be parallelized*, it has been noticed that most of the execution times of a simulation is due to the execution of these methods, i.e., $\frac{t_{total}}{t_{methods}} = 93.2\%$ for 140 neurons, increasing quickly to 99.3% for 240 neurons. This shows the high parallelizability of P-DEVS simulations. Besides, it has also been noticed that most of the execution time is due to the execution of atomic output and transition functions, i.e., $\frac{t_{total}}{t_{trans}+t_{out}} = 91.51\%$ for 140 neurons increasing quickly to 99.08% for 240 neurons.

Figure 7 presents the speed-up obtained for different sizes of networks according to different numbers of threads (implemented in a pool[6]). Each replication has been replicated 30 times leading to a total number of $19 \times 30 \times 4 = 2280$ simulations. It can be seen that in each simulation, the speed-up reaches a maximum which remains constant (cf. Figure 7.c and Figure 7 .d) or decreases (cf. Figure 7.a and Figure 7.b).

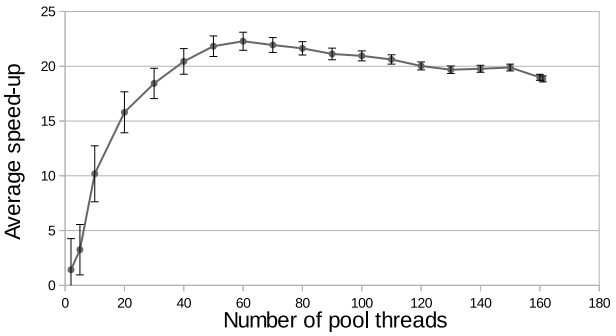
[6]Notice that for each simulation the Java Virtual Machine added also 16 threads for garbage collection and specific to the libraries used in the simulator.

Each best average speed-up obtained in Figure 7 is presented in Figure 8. The optimal number of pool threads is: 20 for 140 neurons, 60 for 240 neurons, 100 for 340 neurons and 50 for 440 neurons. Increasing the number of neurons the average best speed-up decreases and a practical maximum speed-up of 23.5 is achieved. This suggests that the simulations are memory bound, i.e., increasing the number of threads leads to a bottleneck memory access. Further practical investigations are required now to confirm this assumption.

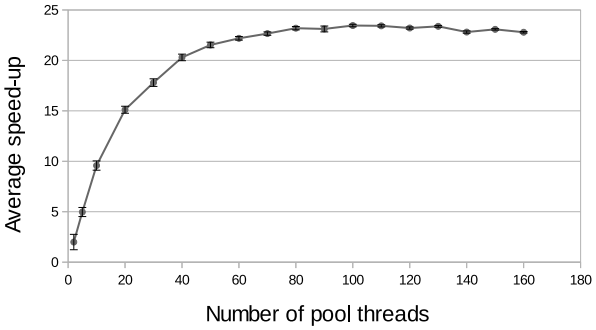
Finally, to investigate the parallelizability of our simulation model, let's consider Amdahl's law [1] as $S(n) = \frac{1}{\tau_{seq}+\frac{1}{n}(1-\tau_{seq})}$ with the *maximum theoretical speed up* $S(n)$ (considering no parallelization overhead) for a *number of threads* n, and the f*raction of total execution time as strictly sequential* as $\tau_{seq} = \frac{t_{seq}}{t_{total}}$. Having $n = 80$ physical cores on the SMP machine used, for 140 neurons, the theoretical maximum speed-up is $S(80) = 14.3$ (while the practical speed-up is 5.14) and for 240 neurons, the theoretical maximum speed-up is $S(80) = 53$ (while the practical maximum speed-up is 22.2). Practical maximum speed-up is less than half of theoretical maximum speed-up, suggesting great further potential speed-up.



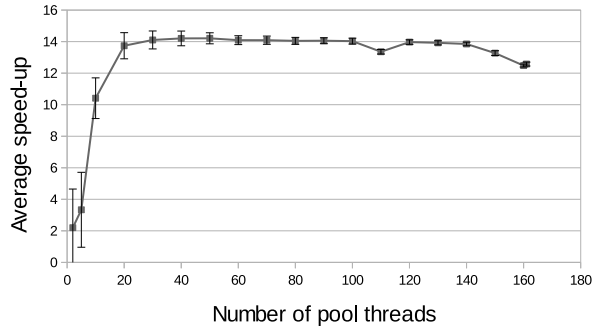
(a)



(b)



(c)



(d)

Figure 7. Comparison of execution time results for an increasing number of pool threads and: (a) 140 neurons, (b) 240 neurons, (c) 340 neurons, and (d) 440 neurons.

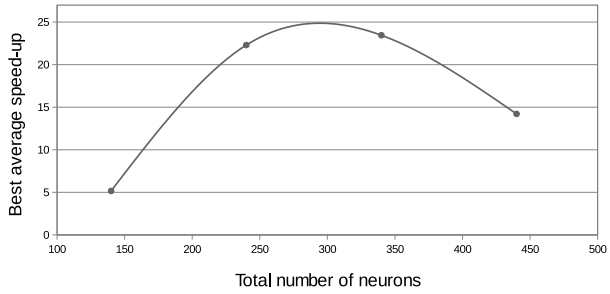


Figure 8. Best average execution-time speed-up for each total number of neurons.

VI. Conclusion and perspectives

This article presented a first formal bridge between computational discrete event systems and networks of spiking neurons. Parallel and stochastic aspects (and their relationship) have been defined explicitly. In P-DEVS a simple way of parallelizing simulations and a link between P-DEVS and (pseudo)random graphs/generators/variables have been proposed. Finally all these structures have been applied to a network of spiking neurons. From a simulation point of view, it can be seen that most of the sequential execution times (more than 90%) can be reduced theoretically. In practice, the simplicity obtained by centralizing most of the computations at the same place requires a strong optimization at software level and a suitable solution at hardware level.

In conclusion, although further technical investigations need to be achieved, it is believed that: the formal structures provided here allow mathematical reasoning at (computational) system level and that the simplicity of the parallel implementation technique should allow further (more efficient) parallelization developments, based on our theoretical maximum speed-up results.

Acknowledgements

Many thanks to Gaëtan Eyheramono and specially to Antoine Dufaure who achieved a first version of the multithreaded implementation. This work has been partially funded by a contract Projets Exploratoires Pluridisciplinaires Bio-Maths-Info (PEPS-BMI 2012), entitled Neuroconf, and funded by Centre National de la Recherche Scientifique (CNRS), Institut national de recherche en informatique et en automatique (INRIA) and Institut National de la Santé et de la Recherche Médicale (INSERM).

References

[1] Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (New York, NY, USA, 1967), AFIPS '67 (Spring), ACM, pp. 483–485.

[2] B. P. Zeigler, T. G. Kim, H. P. *Theory of Modeling and Simulation*. Academic Press, 2000.

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

[4] Castro, R., Kofman, E., and Wainer, G. A formal framework for stochastic discrete event system specification modeling and simulation. *Simulation 86*, 10 (2010), 587–611.

[5] Hill, D. Parallel random numbers, simulation, and reproducible research. *Computing in Science Engineering 17*, 4 (July 2015), 66–71.

[6] Zeigler, B. P. *Theory of Modeling and Simulation*. Wiley, 1976.

[7] Zenke, F., and Gerstner, W. Limits to high-speed simulations of spiking neural networks using general-purpose computers. *Frontiers in Neuroinformatics 8*, 76 (2014).