International Journal of Distributed and Parallel Systems (IJDPS) Vol.4, No.1, January 2013

A PERFORMANCE EVALUATION OF A PARALLEL BIOLOGICAL NETWORK MICROCIRCUIT IN NEURON

Salvatore Cuomo[1], Pasquale De Michele[2] and Francesco Piccialli[3]

University of Naples "Federico II" Department of Mathematics and Applications,
Via Cinthia, 80126, Napoli, ITALY.
[1]salvatore.cuomo@unina.it
[2]pasquale.demichele@unina.it
[3]francesco.piccialli@gmail.com

ABSTRACT

A critical issue in biological neural network modelling is the parameter tuning of a model by means ofthe numerical simulations to map a real scenario. This approach requires a huge amount ofcomputational resources to assesses the impact of every model value that, generally, changes thenetwork response. In this paper we analyse the performance of a CA1neural network microcircuit model for pattern recognition. Moreover, we investigate its scalability andbenefits on multicore and on parallel and distributed architectures.

KEYWORDS

Parallel Computing, GPU Computing, Performance Analysis, Programming, Biological Neural Network.

1. INTRODUCTION

Prototyping and developing computational codes of biological networks, in terms of reliability, efficient, and portable building blocks allow to the computational neuroscientists to simulate real cerebral behaviours and to validate theories and experiments. In prior work,we describedseveral mathematical models and software for biological cells modelling [13];softwareare divided into two main categories: the general and the special purpose Problem Solving Environments (PSEs). The widely diffused PSEsareNEURON [8] and GENESIS [14]. These frameworks have large communities where the users collect and maintain databases of published computational models. In this work, we use NERUON to solve problems at several levels of biological details.

Many research papers ([9], [12], [15], [16] and [17])adopt parallel computing and scientific tools to increase the performance of novel neural network models. Generally, they highlight the biological results, without dealing on the computational aspects related to the parameter issue setting or to the simulation strategies.Moreover, they do not report detailed analysis of the parallel and distributed algorithms, which are essential to optimize a network in terms of its scalability and performance. In other words, building a neural network that reproduces a real scenario requires long and deep stepsto set the model parameters through numerical simulations. The validation of a computational model, by tuning different biological parameters, represents a critical issue in terms of memory allocations and computing time. For example, in [13] we have showed that the tuning of the external electrical stimuli and the random synapses of a modified CA1 model [17] requires a full computational time of $T_{tot} \approx 9h$ for a single cell. The adjustment phase increases the simulation timedramatically, when switching from a single cell to a

DOI : 10.5121/ijdps.2013.4102

network. In this work, the starting point is a multi-pattern modification of a CA1 neural network microcircuit model for pattern recognition [9], implemented in NEURON. In our experiments, the set-up of the network requires the storage of about 45GB of data and several days for any simulation. Moreover, the computational occupancy of the resources strongly depends on the model parameter variation. As a consequence, we study the scalability and the performance of the proposed modification, which is a crucial step to develop a network that recognizes patterns in a real scenario. This work is useful to highlight the strengths and weaknesses of the scientific and parallel computing strategies used in the implemented code. We prove that a pivotal role is played by the combined use of high performance computing techniques, including parallel scientific computing, multi-cores programming and GPU programming. Finally, we suggest solutions that may also be adopted to optimize the neural network code and to simulate a biological scenario in a more reliable time.

The paper in organised as follows. In the Section 2 we show our multi-pattern modification of the microcircuit. The Section 3 describes the numerical results and in the Section 4 performance considerations are discussed. Finally, in the Section 5 we report the conclusions.

2. A MULTI-PATTERN MODIFICATION OF A CA1 MICROCIRCUIT

The model in [9] defines a neural network microcircuit model of the *hippocampus*, a brain region that is involved in the intermediate term storage of the memories that can be consciously recalled: the *declarative memories* [1, 2, 3]. The hippocampus contains two main types of cells: principal excitatory neurons that are the main information processors of this region, and a large kind of inhibitory inter-neurons that form connections locally [5, 6]. In [7] it is proposed an hippocampus feature, the *theta rhythm* (*4-7~ Hz*), that contributes to the memory formation by separating encoding (storage) and retrieval (recall) of memories into different functional half-cycles. In particular, the model in [9] simulates in NEURON the firing time of different hippocampal cell types relative to the theta rhythm in anaesthetised animals, and addresses the roles played by the various types of inhibitory inter-neurons in the dynamical CA1 information processing.

Our modification consists of re-designing the model in [9] by introducing a new multi-pattern recognition strategy. Given a set of *N* patterns, the model is able to **store** all the *N* patterns with the Spike-timing-dependent plasticity (STDP) learning rule and subsequently **recall** these. STDP is a process that adjusts the strength of the connections between neurons in the brain. In STDP rule synapses increase (or decrease) their efficacy if the pre-synaptic spike arrives before (or after) the post-synaptic neuron is activated. For biological aims it is needed to investigate the CA1 network, by means of the recall stage of a large numbers of stored patterns. The complete knowledge of the neuronal phenomena requires the increasing of the patterns and multiple parametric simulations on the network.

As showed in the Figure 1, the model consists of 235 cells: 100 Pyramidal (P), 2 Basket (B), 1 BiStratified (BS), 1 Axo-Axonic (AA) and 1 OriensLacunosum-Moleculare (OLM), that are cells with biophysical properties adapted from literature; 100 CA3, 20 Entorinal Cortex (EC) and 10 medial SEPtum (SEP), that are cells whose behaviour is simulated by means of electrical stimuli. These cells are connected as showed in Table 1. The duration of a single theta rhythm in the network is fixed to *250 ms*: *125 ms* for storage and *125 ms* for recall phases. Repeating the theta cycles for a fixed number of times (*T*) an alternation between phases of the storage and recall occurs. The algorithm is divided in two sub-algorithms: the storage and the recall.

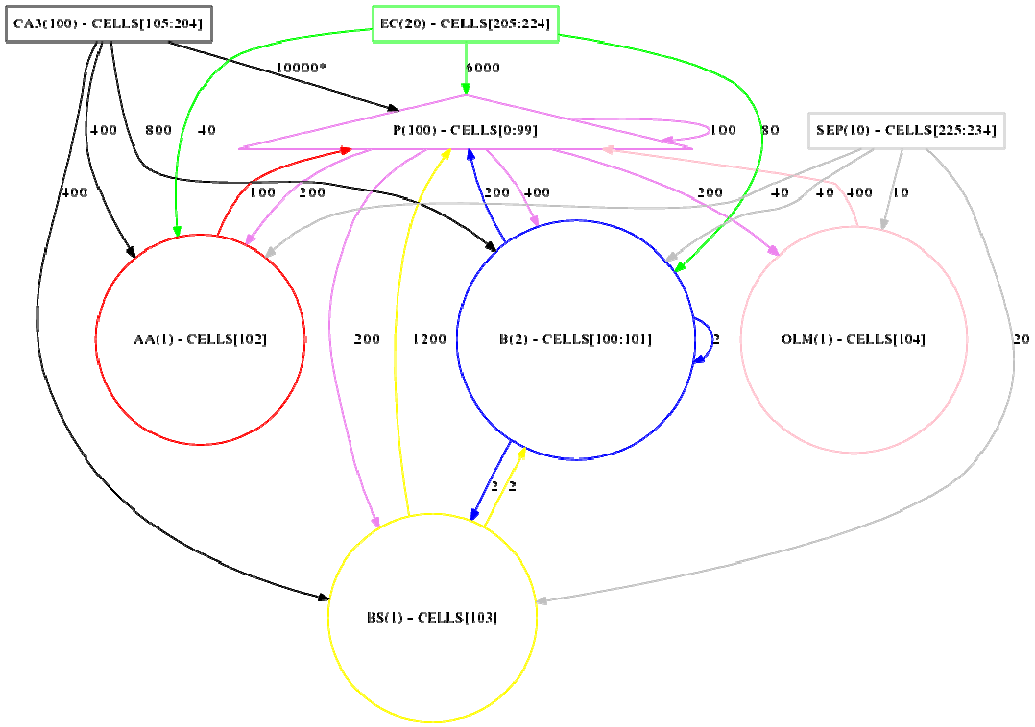

Figure 1. Network topology

2.1. Storage Algorithm

In the Figure 2 (on the left) the graphical representation of the multi-pattern Algorithm 1 is showed. This algorithm for the storage of *N* patterns consists of many functionally dependent tasks.

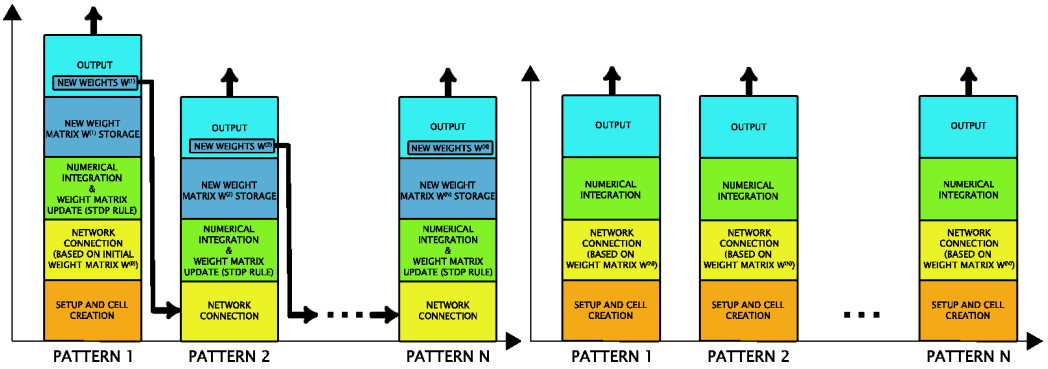

Figure 2. Graphical representations of the multi-pattern algorithms for the storage (on the left) and the recall (on the right) of *N* patterns

Table 1. Network connections

Synapse	Number of sources	(a) Number of targets	(b) Number of connections	(c) Number of synapses in a single connection	Number of synapses in total = ((a)×(b))×(c)
EC→B	20	2	20	2	(2×20)×2=80
EC→AA	20	1	20	2	(1×20)×2=40
EC→P	20	100	20	3	(100×20)×3=6000
CA3→B	100	2	100	4	(2×100)×4=800
CA3→AA	100	1	100	4	(1×100)×4=400
CA3→BS	100	1	100	4	(1×100)×4=400
CA3→P	100	100	100	1	(100×100)×1=10000
SEP→B	10	2	10	2	(2×10)×2=40
SEP→AA	10	1	10	2	(1×10)×2=20
SEP→BS	10	1	10	2	(1×10)×2=20
SEP→OLM	10	1	10	1	(1×10)×1=10
P→P	100	100	1	1	(100×1)×1=100
P→B	100	2	100	2	(2×100)×2=400
P→AA	100	1	100	2	(1×100)×2=200
P→BS	100	1	100	2	(1×100)×2=200
P→OLM	100	1	100	2	(1×100)×2=200
B→P	2	100	2	1	(100×2)×1=200
B→B	2	2	1	1	(2×1)×1=2
B→BS	2	1	2	1	(1×2)×1=2
AA→P	1	100	1	1	(100×1)×1=100
BS→P	1	100	1	12	(100×1)×12=1200
BS→B	1	2	1	1	(2×1)×1=2
OLM→P	1	100	1	4	(100×1)×4=400

initialize $W^{(0)}$
SETUP AND CELL CREATION
for $i = 1$ **to** N **do**
  NETWORK CONNECTION based on $W^{(i-1)}$
  NUMERICAL INTEGRATION and
  WEIGHT MATRIX $W^{(i)}$ UPDATE with STDP rule
  NEW WEIGHT MATRIX $W^{(i)}$ STORAGE
  OUTPUT
end for

Algorithm 1. Multi-patterns storage algorithm

The tasks of a single pattern are organized in stacks, with different execution steps. In detail, these steps are: 1) the network initializing (for the first pattern only); 2) the setup of thenetwork connections by means of a suitable weight matrix **W,** computed with STDP rule; 3) the numerical integration and the **W**update; 4) the storage of **W**; 5) the output generation. The numerical integration consists of the solutions of systems of Ordinary Differential Equations (ODEs) that models the cell computational behaviour. A detailed description of the model is in the Appendix.

The **W** matrix covers a key role in the adaptive multi-pattern recognition, because it is used to opportunely calibrate the network connections, taking into account the synaptic plasticity. In particular, for each pattern i ($1 \leq i \leq N$), the model returns:

$$\mathbf{W}^{(i)} = s(\mathbf{W}^{(i-1)}, \mathbf{c}^{(i)}, \mathbf{p}^{(i)})$$

where$\mathbf{W}^{(i-1)}$ is the weight matrix obtained by the (i-1)-th pattern storage ($\mathbf{W}^{(0)} = 0$), $\mathbf{c}^{(i)}$ is a network connection vector, $\mathbf{p}^{(i)}$ is the i-th pattern to be stored, and s is a function that represents the STDP rule.

The stacks of the storage algorithm are functionally dependent, and our implementation strategy consists of parallelizing the network activity on a single stack (pattern). By using a *round-robin* strategy, the network cells are distributed among the available hosts [10], as showed in Figure 3.

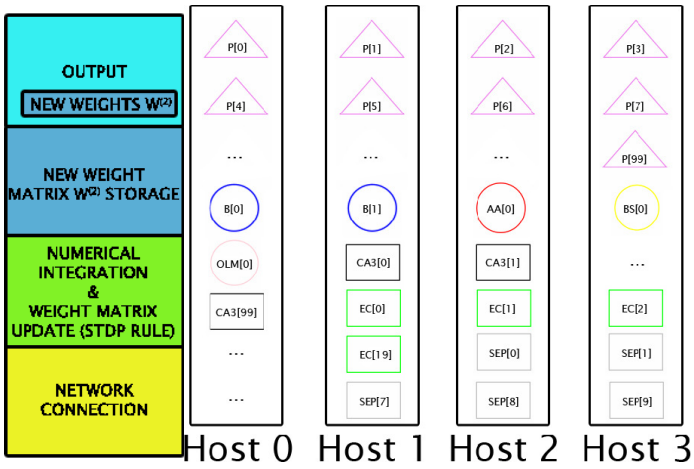

Figure 3. Example of parallelization with 4 available hosts, for a generic execution stack

From parallel computing point of view, the strategy is the following:

(1) a processor integrates the computational modelon its assigned group of cells;

(2) if a spike event occurs between source and target cells, a network communication (`NetCon`)is activated;

(3) theweight matrix information are collected in parallel with a "compare and exchange" strategy.

The parallel performance of the storage algorithm will be discussed in the Section 3.

2.2. Recall Algorithm

In Figure 2 (on the right), the graphical representation of the recall Algorithm 2 is showed.At the *i*-th pattern, the model returns:

$$OUT^{(i)} = r(\mathbf{W}^{(N)}, \mathbf{c}^{(i)}, \mathbf{p}^{(i)})$$

where$\mathbf{W}^{(N)}$ is the weight matrix obtained from the previously storage of the N patterns, $\mathbf{c}^{(i)}$ is a network connection vector, $\mathbf{p}^{(i)}$ is the pattern to be recalled, r is a function that evaluates the recall, $OUT^{(i)}$ is the biological output related on recall performance.

```
Distribute the N patterns on N hosts
for all pattern do
   SETUP AND CELL CREATION
   NETWORK CONNECTION based on W^(N)
   NUMERICAL INTEGRATION
   OUTPUT
end for
```

Algorithm 2. Multi-patterns recall algorithm

During the recall of the N patterns, all stacks are independent, thus it is possible both to distribute and parallelize the execution of these. In detail, a single pattern, that will be recalled, is parallelized in the same way of the storage phase. Let be $K = N * M$ the number of the available hosts on the parallel and distributed architecture,in a first phase N master hosts are selected in order to recall a stored pattern. In a second phase each master host sets the network on M assigned hosts and carries out the recall algorithm, i.e. it executes a stack. For example, suppose we have to recall 10 patterns on a parallel architecture with 8 core hosts. Each host recalls a stored pattern and executes the stack on its 8 core blade.

In Figure 4, the output of the microcircuit for one pattern of the recall algorithm is showed. The simulation is carried out on *6100 ms* with 24 theta cycles and shows the input spikes, the recall quality, the number of spikes of pyramidal cell and the pyramidal cells involved in the synaptic activity. After a phase of network setup and calibration, we point out that the matrix of the weight **W** obtained by the storage algorithm, is used to recall a fixed memory pattern with the quality showed in Figure 4.

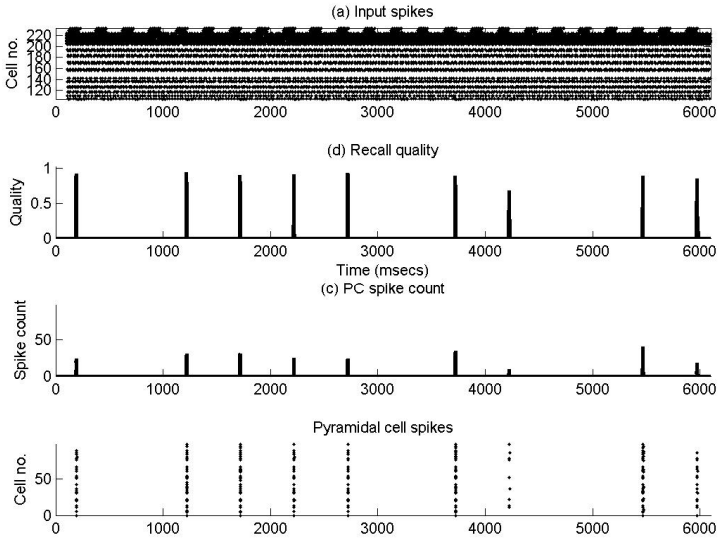

Figure 4. Biological output carried out after a pattern recall during *6100 ms*

In this work we do not discuss about the biological results of the computational neural network, but we remark that a simulation on a workstation with 8 cores requires, at this time, ~910 seconds. This experiment is reported with the aim to show that it is needed to repeat the simulations by tuning the network parameters. Using the network with a large number of patterns and theta cycles, it is a preliminary step for validating the CA1 microcircuit from a biological standpoint. In the next Section we discuss the computational performance of these algorithms.

3. MULTI-CORE AND PARALLEL DISTRIBUTED CONTEXT COMPARISONS

The model is implemented by using the problem solving scientific environment NEURON (v.7.1). The performance tests were carried out on the S.Co.P.E. Grid Infrastructure of University of Naples "Federico II", that consists of 300 blade servers (nodes), each of which with 8 cores "Intel Xeon E5410" at 2.33 GHz (2400 cores in total), connected with several kinds of links: Infiniband, Fiber Channel, 1Gb and 10Gb Ethernet links. Storage and recall algorithms are performed on a set of 10 patterns, both characterized by duration of each theta-cycle of *250ms* and an initial delay of *100ms*. The storage is performed on two different architectures: multicore and distributed.

In order to evaluate the storage and recall performance on these architectures, different parameters are taken into account. The **Runtime** is the total execution time for the application; **Wait** is the time spent for exchanging spikes during a simulation; **Step** is the time spent to integrate ODE systems, checking thresholds, and delivering events (it corresponds to the amount of time needed to numerical integration and weight matrix **W** updating); **Setup and cell creation** is the time to setupbiological parameters and to create all cells of the network; **Network connection** is the time to setup network connection; **New weight matrix storage** is the time to collect, from each host, the values of the weight matrix and storing these for the next pattern to be processed; **Output** is the time to store biological output information; **Others** is the time to collect non-functional information.

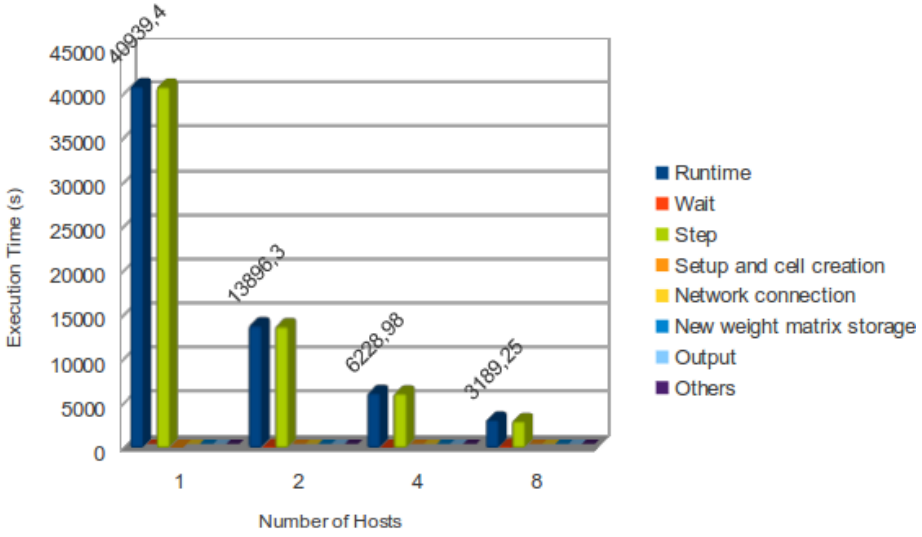

Figure 5. Execution times for the storage of 10 patterns with 8 theta cycles on a multicore architecture

The Figure 5 shows that for the storage phase, on a multicore architecture, increasing the number of hosts (from 1 to 8) leads to a huge reduction of **Runtime**, substantially equivalent to the **Step** time. In fact, from the Figure 6, it is possible to see a reduction of**Step** from the *100%* to the *95,47%* of the **Runtime**, moving from 1 to 8 hosts. Hence, the other execution steps (setup and cell creation, network connection, weight matrix storage and output generation) do not affect the overall time.

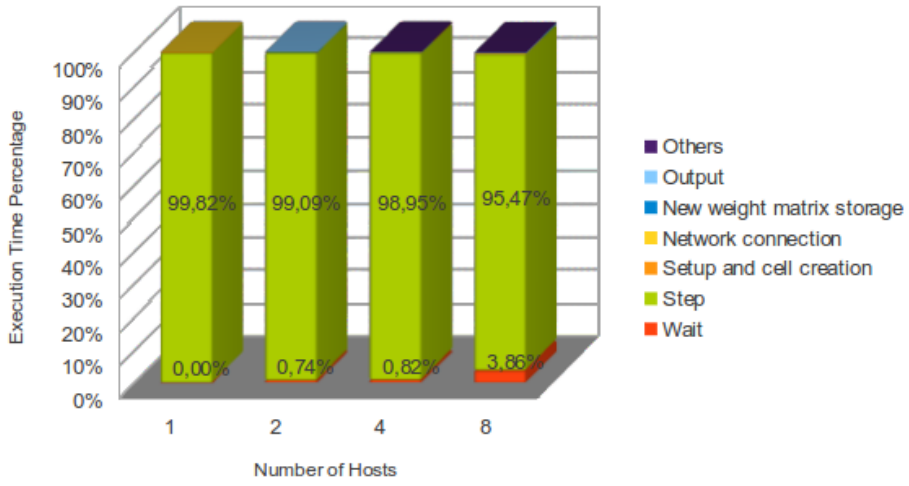

Figure 6. **Runtime** percentages for the storage of 10 patterns with 8 theta cycles on a multicore architecture

Moreover, the Figure 6 shows a slight increment of the **Wait** time (from *0%* to *3,86%* of the **Runtime**) due to the growing number of communications among the cells mapped on different hosts. This phenomenon is amplified when moving from a multicore to a distributed architecture.

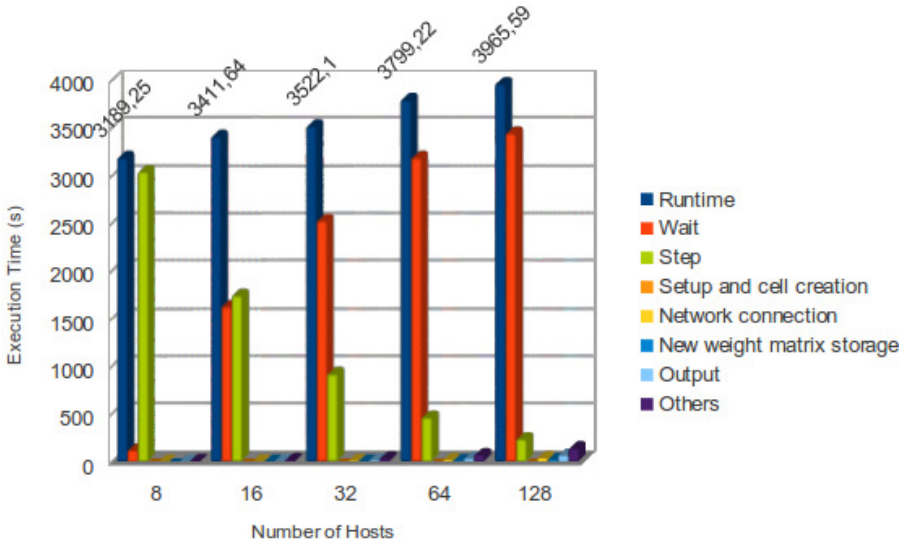

Figure 7. Execution times for the storage of 10 patterns with 8 theta cycles on a distributed architecture

The Figure 7 shows that on a distributed architecture a slight worsening of the performance occurs. In detail, the time needed for the numerical integration (**Step**) continues to decrease with doubling of the host number and there is a strong increment of the **Wait**time (the amount of time needed to synchronize the communication between two cells mapped on two different hosts). The Figure 8 confirms this: with 16 hosts (distributed on 2 different nodes) **Wait** time reaches the *47,76%* of the **Runtime**, while with 128 hosts it further increases, reaching the *86,93%* of the **Runtime**.

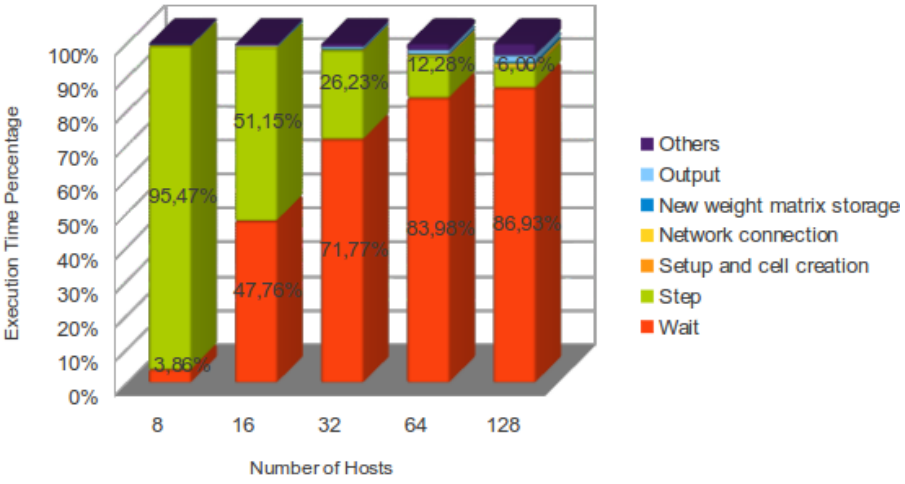

Figure 8. **Runtime** percentages for the storage of 10 patterns with 8 theta cycles on a distributed architecture

Moreover, this behaviour is particularly clear by observing the Figure 9. NEURON uses an event delivery system to implement spike-triggered synaptic transmission between cells. Detection of a spike launches an event that, after an appropriate delay, will be delivered to a

target cell. The basic problem that has to be overcome in a parallel simulation environment is that the source cell and its target usually do not exist on the same host [10]. Hence, it is evident that, moving from a multicore to a distributed architecture, the time for synchronize different hosts rises with increasing of the involved host number.

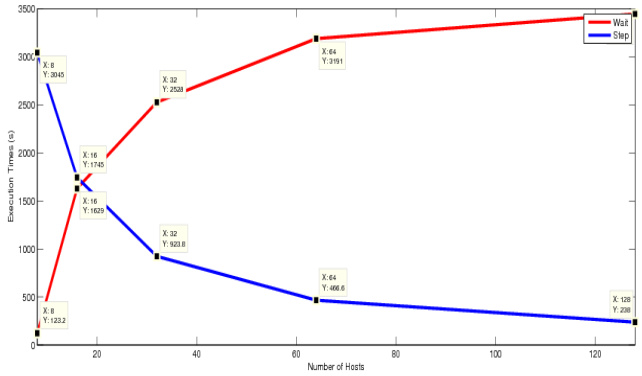

Figure 9. **Step** and **Wait** curves with the increasing of number of hosts

On a multicore architecture, the recall algorithm has the same performance we have discussed for the storage phase. On the other hand, as we mentioned earlier, it is possible to implement the recall algorithm on a distributed architecture: in this way, each algorithm performs the recall of a single pattern on a node of the architecture. A typical parametric execution of a distributed recall phase is characterized by the pair (*Number of pattern*, *Number of theta cycles*). In the Table 2 the performance of the parametric execution (*10 patterns*, *16 theta cycles*) are showed. The recall of *N* patterns is performed in *~600s*, that is the time needed to recall a single pattern only, obtaining a substantial performance improvement: without the distribution of the algorithm, the **Runtime** would have been equal to *~6000s*.

Table 2. Execution times for the recall of 10 patterns with 16 theta cycles, distributing the model

Pattern ID	Runtime	Wait	Step	#Spikes	Set and Connect	Output
1	608,775	8,71033	597,987	7975	0,54125	0,61625
2	608,861	8,93715	598,457	8149	0,54125	0,6375
3	607,072	8,51312	597,083	8061	0,54125	0,64875
4	607,812	8,67061	597,683	8033	0,54125	0,635
5	607,119	9,37756	596,286	7946	0,54125	0,6275
6	607,035	8,71243	596,895	7992	0,54125	0,6325
7	607,726	8,79549	597,473	8030	0,54125	0,62125
8	607,191	8,39568	597,321	7848	0,54125	0,63875
9	608,27	10,0982	596,692	8022	0,54125	0,63625
10	608,42	9,3668	597,576	8046	0,54125	0,6325

4. PERFORMANCE CONSIDERATIONS

In the Section 3, we have discussed the increase of the **Wait** time when moving from a multicore to a distributed architecture. Now, we investigate how to limit the **Wait** time growing. Better performance and containment of **Wait** time are given on a multicore architecture. This kind of architecture is restricted by technological limits related to the small number of cores that can work together (8-16 cores at this time). Consequently, we need to find an architecture incorporating the idea behind the multicore computing. The Graphical Processing Unit (GPU) computing is the solution to this problem. NEURON is a simulator that incorporates molecular, detailed compartmental modified Hodgkin-Huxley models of axons and dendrites from anatomical observations, and various ion channels to biophysical details, but unfortunately, at this time, it is not compatible with parallelization on GPUs. Accordingly, while it is biologically accurate, it incurs tremendous computational costs for simulation [11]. On the other hand, there exist many simulation environments that implement and simulate biological neural networks on GPU architectures.

In [11] is described an easy to use simulator that provide significant computational performance, as it is able to run on NVIDIA GPUs. Despite this software, differently from NEURON, uses theIzhikevich neuron model [18] and does not allow to specify the cell morphologies with great accuracy from a biological point of view,here we implement a simplified "ad-hoc" version of our model with this tool.

Firstly, we created all the cells specified in the NEURON version.The Figure 10 shows the creation of Pyramidal and Basket cells, in NEURON (on the left) and in the "ad-hoc" (on the right) versions. Then, we set the connection between the cells, reproducing the same network topology specified with NEURON. In Figure 11 is showed the connection between Pyramidal (pre-synaptic) and Basket Cells (post-synaptic), implemented in NEURON (on the left) and in the "ad-hoc" (on the right) versions. Finally, we simulated the network activity, on CPU and GPU architectures.

We compared the performance between CPU(2.4GHz quad-core "Intel Xeon E5620") and GPU (NVIDIA Tesla S2050) versions, and we observed a reduction of *85%* of the execution time, moving from CPU to GPU, as showed in Figure 12. This behaviour is due to the synchronization time removal.

```
/*Cell creation - NEURON version*/
// Number of Pyramidal Cells
N_P    = 100
// Number of Basket Cells
N_B    = 2
// Starting resting potential
vinit = -65
...
for (i=pc.id; i < ntot; i += pc.nhost) {
  if (i < N_P) {
    // Pyramidal Cell creation
    cell = new PyramidalCell(redsAHP, redmAHP, vinit,
          tauca, thna3, thna3dend, somakap, somakad,
          ghsoma, somacaT, somacaL, shift)
  } else if (i < N_P + N_B) {
    // Basket Cell creation
    cell = new BasketCell()
  }
  ...
  cells.append(cell)
  ...
}
```

```
/*Cell creation - Ad-hoc version*/
// Number of Pyramidal Cells
#define N_P    100
// Number of Basket Cells
#define N_B    2
...
// Starting resting potential
float vinit=-65.0f;
// Pyramidal Cell creation
int pCells=s.createGroup("excit", N_P, EXCITATORY_NEURON);
s.setNeuronParameters(pCells, 0.02f, 0.2f, vinit, 8.0f);
// Basket Cell creation
int bCells=s.createGroup("inhib", N_B, INHIBITORY_NEURON);
s.setNeuronParameters(bCells, 0.1f,  0.2f, vinit, 2.0f);
...
```

Figure 10. Cell creation in the NEURON and the "ad-hoc" versions

```
/*Cell connection - NEURON version*/
// Number of Pyramidal Cells
N_P                 = 100
// Number of Basket Cells
N_B                 = 2
// Index of the first Pyramidal Cell
iPC                 = 0
// Index of the first Basket Cell
iBC                 = N_P
// Number of connections received
// by each Basket Cell from PCs (excit)
PC_BC               = N_P
// Number of synapses in a single connection
N_SYN_PC_BC         = 2
// Weight of the connections
Pcell2Bcell_weight = 0.0005
// Delay of the connections
Pcell2Bcell_delay  = 1
...
// Connection between Pyramidal Cells and Basket Cells:
  // Total Number of synapses: N_SYN_PC_BC * N_P * N_B = 400.
connectcells(N_B, iBC, N_P, iPC, PC_BC, 1, N_SYN_PC_BC,
             Pcell2Bcell_delay, Pcell2Bcell_weight)
...
```

```
/*Cell connection - Ad-hoc version*/
// Number of Pyramidal Cells
#define N_P                 100
// Number of Basket Cells
#define N_B                 2
// Number of synapses in a single connection
#define N_SYN_PC_BC         2
// Delay of the connections
#define Pcell2Bcell_weight  0.0005
// Delay of the connections
#define Pcell2Bcell_delay   1
// connection probability (1 = 100%)
#define ALL_TO_ALL          1
...
// Connection between Pyramidal Cells and Basket Cells:
  // Total Number of synapses: N_SYN_PC_BC * N_P * N_B = 400
for (int i=1; i <= N_SYN_PC_BC ; i++)
  s.connect(pCells,bCells,"full", Pcell2Bcell_weight,
           Pcell2Bcell_weight, ALL_TO_ALL, Pcell2Bcell_delay,
           Pcell2Bcell_delay, SYN_FIXED);
...
```

Figure 11. Example of cell connection in the NEURON and the "ad-hoc" versions: each Basket Cell has 2 synapses with each Pyramidal Cell; 400 synapses in total

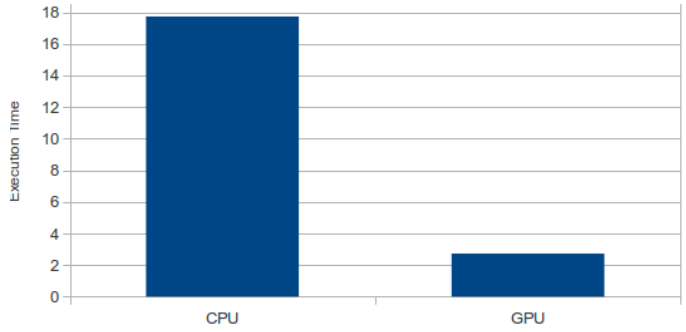

Figure 12. Execution time reduction moving from a CPU to a GPU architecture

5. CONCLUSIONS

The simulation of biological neural networks is a challenging application from a computational point of view. The calibration and setup of a network require mathematical models in order to simulate the biological behaviour of the different cell type and sophisticated programming environments for developing simulation codes. In practice, building a microcircuit that mimes the real behaviour of a biological neural network, with a large number of connections between its neurons, requires algorithms and communication strategies computationally expensive.

In this work we propose a modification of an exiting CA1 microcircuit for the multi-pattern recognition. Our aim is to analyze the performance of the proposed code in a parallel, a distributed and a multicore architecture. We observe that the main problem that has to be overcome is the communication between a source cell and its target, in a parallel simulation environment. We proof that the microcircuit well scale on a multicore architecture, when the spike communications are carried out on a dedicated bus. We strongly suggest to use general-purpose simulation environments that support massively parallel multicore programming. Finally, we think that the performance analysis of the proposed microcircuit is useful in order to simulate a large number of microbiological multi-pattern recognition experiments in an acceptable computing time. We are working on the microcircuit optimization on a massively parallel GPU architecture. We are investigating the compatibility of NEURON with GPU

computing and exporting the code of our microcircuit on a more general-purpose simulator that supports the GPU.

ACKNOWLEDGEMENTS

This paper is financial supported by Multicentre Research Project funded by Compagnia di San Paolo: "Molecular Mechanisms of Memory: Identification and Modelling".

A. APPENDIX: MATHEMATICAL FORMALISM

A.1. Pyramidal Cells

The somatic (s), axonic (a) and radiatum (rad), lacunosum-moleculare (LM) and oriens (ori) dendritic compartments obey the following current balance equations:

$$C\frac{dV_s}{dt} = -I_L - I_{Na} - I_{kdr} - I_A - I_M - I_h - I_{sAHP} - I_{mAHP} - I_{CaL} - I_{CaT} - I_{CaR} - I_{buff} - I_{syn} \quad (1)$$

$$C\frac{dV_a}{dt} = -I_L - I_{Na} - I_{kdr} - I_M - I_{syn} \quad (2)$$

$$C\frac{dV_{rad,\ ori}}{dt} = -I_L - I_{Na} - I_{kdr} - I_A - I_M - I_h - I_{sAHP} - I_{mAHP} - I_{CaL} - I_{CaT} - I_{CaR} - I_{buff} - I_{syn} \quad (3)$$

$$C\frac{dV_{LM}}{dt} = -I_L - I_{Na} - I_{kdr} - I_A - I_{syn} \quad (4)$$

whereI_L is the leak current, I_{Na} is the fast sodium current, I_{kdr} is the delayed rectifier potassium current, I_A is the A-type K^+ current, I_M is the M-type K^+ current, I_h is a hyperpolarizing h-type current, I_{CaL}, I_{CaT} and I_{CaR} are the L-, T- and R-type Ca^{2+} currents, respectively, I_{sAHP} and I_{mAHP} are slow and medium Ca^{2+} activated K^+ currents, I_{buff} is a calcium pump/buffering mechanism and I_{syn} is the synaptic current.

The sodium current is described by:

$$I_{Na} = \bar{g}_{Na} \times m^2 \times h \times s \times (V - E_{Na}) \quad (5)$$

The delayed rectifier current is given by:

$$I_{Kdr} = \bar{g}_{Kdr} \times m^2 \times (V - E_K) \quad (6)$$

The fast inactivating A-type K^+ current is described by:

$$I_A = \bar{g}_A \times \bar{n}_A \times (V - E_K) \quad (7)$$

The hyperpolarizing h-current is given by:

$$I_h = g_h \times tt \times (V - E_h) \quad (8)$$

The slowly activating voltage-dependent potassium current, I_M, is given by the equation:

$$I_m = 10^{-4} \times T_{adj}(°C) \times \bar{g}_m \times m \times (V - E_K) \quad (9)$$

The slow after-hyperpolarizing current, I_{sAHP}, is given by:

$$I_{sAHP} = \bar{g}_{sAHP} \times m^3 \times (V - E_K) \quad (10)$$

The medium after-hyperpolarizing current, I_{mAHP}, is given by:

$$I_{mAHP} = \bar{g}_{mAHP} \times m \times (V - E_K) \tag{11}$$

The somatic high-voltage activated (HVA) L-type Ca^{2+} current is given by:

$$I_{CaL}^{S} = \bar{g}_{CaL}^{S} \times m \times \frac{0.001mM}{0.001mM + ca_{in}} \times ghk(V, ca_{in}, ca_{out}) \tag{12}$$

where the dendritic L-type calcium channels have different kinetics:

$$I_{CaL}^{d} = \bar{g}_{CaL}^{d} \times m^3 \times h \times (V - E_{Ca}) \tag{13}$$

The low-voltage activated (LVA) T-type Ca^{2+} channel kinetics are given by:

$$I_{CaT} = \bar{g}_{CaT} \times m^2 \times h \times \frac{0.001mM}{0.001mM + ca_{in}} \times ghk(V, ca_{in}, ca_{out}) \tag{14}$$

The HVA R-type Ca^{2+} current is described by:

$$I_{CaR} = \bar{g}_{CaR} \times m^3 \times h \times (V - E_{Ca}) \tag{15}$$

Finally, a calcium pump/buffering mechanism is inserted at the cell body and along the apical and basal trunk. The factor for Ca^{2+} entry was changed from $f_e=10000$ to $f_e=10000/18$ and the rate of calcium removal was made seven times faster.

The kinetic equations are given by:

$$drive\_channel = \begin{cases} -f^e \times \dfrac{I_{Ca}}{0.2 \times FARADAY} & if\ drive\_channel > 0\ mM/ms \\ 0 & otherwise \end{cases}$$

$$\frac{dca}{dt} = drive\_channel + \frac{(10^{-4}(mM) - ca)}{7 \times 200(ms)}. \tag{16}$$

A.2. Axo-axonic, Basket and Bistratified Cells

All compartments obey the following current balance equation:

$$C\frac{dV}{dt} = -I_{ext} - I_L - I_{Na} - I_{Kdr,\,fast} - I_A - I_{CaL} - I_{CaN} - I_{AHP} - I_C - I_{syn} \tag{17}$$

where$C$ is the membrane capacitance, V is the membrane potential, I_L is the leak current, I_{Na} is the sodium current, $I_{Kdr,fast}$ is the fast delayed rectifier K^+ current, I_A is the A-type K^+ current, I_{CaL} is the L-type Ca^{2+} current, I_{CaN} is the N-type Ca^{2+} current, I_{AHP} is the Ca^{2+}-dependent K^+ (SK) current, I_C is the Ca^{2+} and voltage-dependent K^+ (BK) current and I_{syn} is the synaptic current.

The sodium current is described by:

$$I_{Na} = g_{Na} m^3 h (V - E_{Na}) \tag{18}$$

The fast delayed rectifier K^+ current, $I_{Kdr,fast}$ is given by:

$$I_{Kdr,\,fast} = g_{Kdr,\,fast} n_f^4 (V - E_K) \tag{19}$$

The N-type Ca^{2+} current, I_{CaN}, is given by:

$$I_{CaN} = g_{CaN} c^2 d(V - E_{Ca}) \tag{20}$$

The Ca^{2+}-dependent K^+ (SK) current, I_{AHP}, is described by:

$$I_{AHP} = g_{AHP} q^2 (V - E_K) \tag{21}$$

The A-type K^+ current, I_A, is described by

$$I_A = g_A ab(V - E_K) \tag{22}$$

The L-type Ca^{2+} current, I_{CaL}, is described by:

$$I_{CaL} = g_{CaL} \times s_\infty^2 \times V \times \frac{1 - \frac{[Ca^{2+}]_i}{[Ca^{2+}]_0} e^{2FV/kT}}{1 - e^{2FV/kT}} \tag{23}$$

whereg_{CaL} is the maximal conductance, s_∞ is the steady-state activation variable, F is Faraday's constant, T is the temperature, k is Boltzmann's constant, $[Ca^{2+}]_0$ is the equilibrium calcium concentration and $[Ca^{2+}]_i$ is described in Eq (25).

A.3. OLM Cell

The somatic (s), axonic (a) and dendritic (d) compartments of each OLM cell obeyed the following current balance equations:

$$C\frac{dV_s}{dt} = -I_{ext} - I_L - I_{Na,s} - I_{K,s} - I_A - I_h - I_{syn} \tag{24}$$

$$C\frac{dV_d}{dt} = -I_{ext} - I_L - I_{Na,d} - I_{K,d} - I_A - I_{syn} \tag{25}$$

$$C\frac{dV_a}{dt} = -I_{ext} - I_L - I_{Na,d} - I_{K,d} \tag{26}$$

The sodium current is described by:

$$I_{Na} = g_{Na} m^3 h(V - E_{Na}) \tag{27}$$

where$m$ and h are the activation and inactivation variables, respectively.

The potassium current, I_K, is described by:

$$I_K = g_K n^4 (V - E_K) \tag{28}$$

where$n$ is the activation variable for this channel.

The transient potassium current, I_A, is described by:

$$I_A = g_A ab(V - E_K) \tag{29}$$

where$a$ and b are the activation and inactivation variables, respectively.

The nonspecific cation channel, I_h, is described by:

$$I_h = g_h r(V - E_r) \tag{30}$$

where *r* is the activation variable for this channel.

A.4. Septal Cells

Septal cell output was modeled as bursts of action potentials using a presynaptic spike generator. A spike train consisted of bursts of action potentials at a mean frequency of *50 Hz* for a half-u cycle (*125 ms*; corresponding to a recall period) followed by a half-u cycle of silence. Due to 40% noise in the interspike intervals, the 10 spike trains in the septal population were asynchronous.

A.5. Entorinal Cortical Cells

EC cells were also modeled as noisy spike trains, using a pre-synaptic spike generator. A spike train consisted of spikes at an average gamma frequency of *40 Hz*, but with individual spike times Gaussian-distributed around the regular ISI of *25 ms*, with a standard deviation of 0.2. The population of EC inputs fired asynchronously.

A.6. CA3 Pyramidal Cells

CA3 pyramidal cells were modeled as spike trains of the same form and with the same characteristics (mean frequency and noise level) as the EC cells. Onset of CA3 firing was delayed by *9 ms* relative to the EC trains to model the respective conduction delays of direct and trisynaptic loop inputs to CA1.

REFERENCES

[1] Eichenbaum, H., Dunchenko, P., Wood, E., Shapiro, M. and Tanila, H. (1999). The hippocampus, memory and place cells: Is it spatial memory or a memory of space? NEURON, 23, 209-226.

[2] Andersen, P., Morris, R., Amaral, D., Bliss, T. and O'Keede, J. (2007).The Hippocampus Book. Oxford: University Press.

[3] Wood, E., Dunchenko, P. and Eichenbaum, H. (1999). The global record of memory in hippocampal neuronal activity.NATURE, 397, 613-616.

[4] Treves, A. and Rolls, E. (1994). Computational analysis of the role of the hippocampus in memory.HIPPOCAMPUS, 4, 374-391.

[5] Freund, T.F. and Buzsaki, G. (1996). Interneurons of the hippocampus.HIPPOCAMPUS, 6, 347-470.

[6] Somogyi, P. and Klausberger, T. (2005). Defined types of cortical interneurons structure space and spike timing in the hippocampus. J PHYSIOL, 562, 9-26.

[7] Hasselmo, M.E., Bodelon, C. and Wyble, B. (2002). A proposed function of the hippocampal theta rhythm: Separate phases of encoding and retrieval of prior learning. NEURAL COMPUT, 14, 793-817.

[8] Carnevale, N.T. and Hines, M.L. (2006). The NEURON book.Cambridge University Press.

[9] Cutsuridis, V., Cobb, S. and Graham, B.P. (2010).Encoding and Retrieval in a Model of the Hippocampal CA1 Microcircuit.HIPPOCAMPUS, 20, 423-446.

[10] Hines, M.L. and Carnevale, N.T. (2008). Translating network models to parallel hardware in NEURON. J NEUROSCI METH, 169(2), 425-455.

[11] Richert, M., Nageswaran, J.M., Dutt N. and Krichmar, J.L. (2011).An efficient simulation environment for modeling large-scale cortical processing. FRONT NEUROINFORM, 5:(19), doi: 10.3389/fninf.2011.00019.

[12] Myers, C.E. and Scharfman, H.E. (2011). Pattern Separation in the Dentate Gyrus: A Role for the CA3 Backprojection, HIPPOCAMPUS, 21:1190-1215.

[13] Cuomo, S., De Michele, P. and Chinnici, M. (2011). Parallel tools and techniques for biological cells modelling, BuletinulInstitutuluiPolitehnic DIN IASI, Automatic Control and Computer Science Section, LVII (LXI), Fasc. 3, 2011, pp. 61-75, ISSN 1200-2169.

[14] Bower, J.M., Beeman, D. *et al*. (2003). The Book of Genesis. Internet Edition.

[15] Cutsuridis, V., Hunter, R., Cobb, S. and Graham, B.P. (2007). Storage and recall in the CA1 microcircuit of the hippocampus: A biophysical model. Vol. 8(suppl 2), 16th Annual Computational Neuroscience Meeting CNS*2007, Toronto, Canada, BMC NEUROSCI, p. 33.

[16] Cutsuridis, V. and Wenneckers, T. (2009).Hippocampus, microcircuits and associative memory. HIPPOCAMPUS 22, 1120–1128.

[17] Bianchi, D., Marasco, A., Limongiello, A., Marchetti, C., Marie, H., Tirozzi, B. and Migliore, M. (2011). On the mechanisms underlying the depolarization block in the spiking dynamics of CA1 pyramidal neurons. J COMPUT NEUROSCI - DOI 10.1007/s10827-012-0383-y.

[18] Izhikevich, E.M. (2004). Which model to use for cortical spiking neurons? IEEE TRANS NEURAL NETW. 15, 1063–1070.

Authors

Salvatore Cuomo is an Assistant Professor at University of Naples “Federico II”.

Pasquale De Michele is a Ph.D.Student in “Computer Science” at University of Naples “Federico II”.

Francesco Piccialli is a Graduated in “Computer Science” at University of Naples “Federico II”.