SEVENTH FRAMEWORK PROGRAMME
Call FP7-ICT-2009-4 Objective ICT-2009.8.1
[FET Proactive 1: Concurrent Tera-device Computing]

Project acronym: **EURETILE**
Project full title: **European Reference Tiled Architecture Experiment**
Grant agreement no.: **247846**

D7.4 – Final DAL release of Data-flow, DPSNN, LQCD Kernels

Lead contractor for deliverable: INFN
Due date of submission: 2014-Sep-30
Revision: See document footer.

Project co-funded by the European Commission within the Seventh Framework Programme	
Dissemination Level: PU	
PU	Public
PP	Restricted to other programme participants (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Final DAL Release of Data-flow kernels, Distributed Plastic polychronous Spiking Neural Net and LQCD

Pier Stanislao Paolucci[1], Iuliana Bacivarov[2], Devendra Rai[2], Lars Schor[2], Lothar Thiele[2], Elena Pastorelli[1]

[1]INFN Roma "Sapienza"
[2]Computer Engineering and Networks Laboratory, ETH Zurich

Authorship notes:

- Corresponding author: Pier Stanislao Paolucci: e-mail: pier.paolucci@roma1.infn.it
- The team of ETH Zurich is author of the "Advanced Benchmarks for Embedded Systems" Chapter.
- The team of INFN is author of the "Brain Simulation Benchmark" Chapter;

Table of Contents

1. Introduction

The benchmarks here described have been used in 2014 to exercise the integrated software tool-chain on both the QUonG hardware experimental platform and the VEP simulated platform. The results of such experiments are described in deliverables D8.3 produced by WP8 (about final Software Tool Chain Integration and measurements) and D6.4 (about final Hardware developments and measurements). Part of such results have been already published in the joint consortium paper [3], other will be published in the EURETILE "final publishable report" and submitted as regular journal paper.

Here, we discuss the final release of a set of application benchmarks developed to experiment the dynamic, many-process software tool-chain on the many-tile fault-tolerant execution platform developed by the EURETILE project (January 2010 – September 2014), see [3,10]. Here, we present also the improvements to the benchmark set produced during the last project period (Jan-Sep 2014).

The ambition of the EURETILE project is to identify common techniques that could be applied to both the Embedded Systems and HPC domains, with a focus on dynamic workload characterized by heavy numerical processing requirements.

This document is the second public deliverable (D7.4) of the EURETILE work-package 7 "Challenging Tiled Applications".

The reader is referred to the previous public deliverable (D7.3, see [9]) for a description of the criteria for the selection of the benchmarks, their structure and coding, and for a description of the benchmark development activities performed until Dec 2013.

The structure of this document is the following: Chapter 2 summarizes the final status of the set of benchmarks for Embedded Systems, while Chapter 3 is about the simulation of large scale plastic spiking neural networks.

2. Advanced Benchmarks for Embedded Systems

In order to wrap-up the development process, we give an overview on all embedded benchmarks developed during the EURETILE project and discuss their impact on the results achieved by the overall project. The benchmarks themselves are available online for download at http://www.dal.ethz.ch. The presented benchmarks are also used in deliverable 3.4 and deliverable 8.3 to evaluate the EURETILE design flow in general and the fault management techniques added to the design flow in particular. To save space, we only report the results in deliverable 3.4 and deliverable 8.3.

All benchmarks developed in this work package have been developed using the distributed application layer (DAL) formalism. DAL is a scenario-based design flow that supports the design, optimization, and simultaneous execution of multiple applications targeting heterogeneous many-core systems. We refer the reader to [1 – 3] in Section 2.5 for more details. In particular, DAL allows specifying applications based on the Kahn process network (KPN) model of computation. In case multiple applications share the system, a finite state machine is elaborated to specify the interactions between the applications. Furthermore, various benchmarks applications have been developed in conjunction with

the semantics of expandable process networks (EPNs) [4, 5]. EPNs extend conventional KPNs by abstracting several possible granularities in a single specification. This enables the application to vary its degree of parallelism if the computational demand of an application changes or if the number of available tiles varies due to applications that are started or stopped.

The remainder of this chapter is organized as follows. First, we review three applications that are typically used in digital signal processing. In Section 2.2, we describe three multimedia applications, namely a multi-stage video processing application, a H.264 codec pair, and a ray-tracing application. In Section 2.3, we describe a distributed array-sorting algorithm. Finally, in Section 2.4, we describe an advanced picture-in-picture (PiP) software for embedded video processing systems.

2.1 DSP/Data-flow Kernels

During 2011 and 2012, various applications have been developed that are typically used in digital signal processing (DSP), namely a distributed implementation of an *N*-point discrete Fourier transformation (DFT), an *N*-order IIR filter, and a distributed implementation of a matrix multiplication. These benchmarks have then been used to evaluate and demonstrate the capabilities of the EURETILE design flow.

2.1.1 Discrete Fourier Transformation (FFT)

The first benchmark application is an *N*-point discrete Fourier transform (DFT). The DFT is a block operation, which takes *N* complex-valued (time-domain) input coefficients and transforms them into *N* complex-valued (frequency-domain) output coefficients. An efficient implementation of the DFT is the decimation-in-time radix-2 FFT. In our implementation, the computation is split up into $N/2 \cdot \log_2(N)$ 2-point FFTs.

2.1.2 N-Order Infinite Impulse Response Filter (IIR)

The *N*-order infinite impulse response filter (IIR) has been implemented by decomposing the *N*-order IIR filter into a decomposition of first order IIR filters.

2.1.3 Matrix Multiplication

The considered *NxN* matrix multiplication implementation splits the computation of the matrix product up into single multiplications and additions. In particular, each process then performs a multiplication followed by an addition. Single coefficients of the resulting matrix are computed by appropriately connecting the processes.

2.2 Multimedia Applications

Strong focus has been given on multimedia applications. In particular, a multi-stage video processing application, an H.264 codec pair, and a ray-tracing application have been implemented using the DAL formalism. All three benchmarks have been successfully applied in various publications (e.g., [1 – 6]) and used to evaluate the concepts proposed in work-package 3 and work-package 8. We refer the interested reader to deliverable 3.4 and deliverable 8.3 for more details.

2.2.1 Multi-Stage Video Processing Application

A multi-stage video-processing application has been implemented that decodes a motion-JPEG video stream and then applies a motion detection method to the decoded video

stream. Figure 2-1 shows the process network of the considered application. The MJPEG decoder can decode multiple video frames in parallel. The motion detection method is composed of a Gaussian blur, a gradient magnitude calculation using Sobel filters, and an optical flow motion analysis. Tokens transmitted between these three components correspond to single video frames, but in all filters, the calculation of an output pixel is independent of the other output pixels so that a high degree of data parallelism can be achieved. Case studies presented, e.g., in [7], have shown that the application is well suited for the execution on modern heterogeneous systems that consist of multi-core CPUs and GPUs.

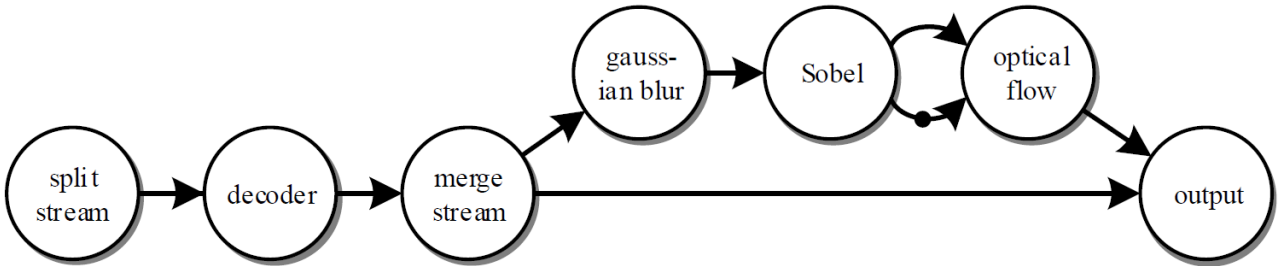

Figure 2-1. Process network of a video-processing application.

2.2.2 H.264 Codec

The H.264/MPEG-4 AVC (Advanced Video Codec) is one of the most widely used video coding standards in recent years. In order to evaluate the effect of intra-application dynamism, we implemented a distributed version of the H.264 encoder and decoder pair using the DAL formalism. Detailed evaluation results based on the H.264 codec benchmark are provided in deliverable 7.3.

The considered implementation of the H.264 codec is based on code for HOPES [8], which has been provided by the Seoul National University and supports the baseline profile of the coding standard. The basic implementation of the encoder is illustrated in Figure 2-2. It uses task division on the macroblock level. In particular, the set of functions is divided among five processes: *Init*, *ME*, *Encode*, *Deblock*, and *VLC*.

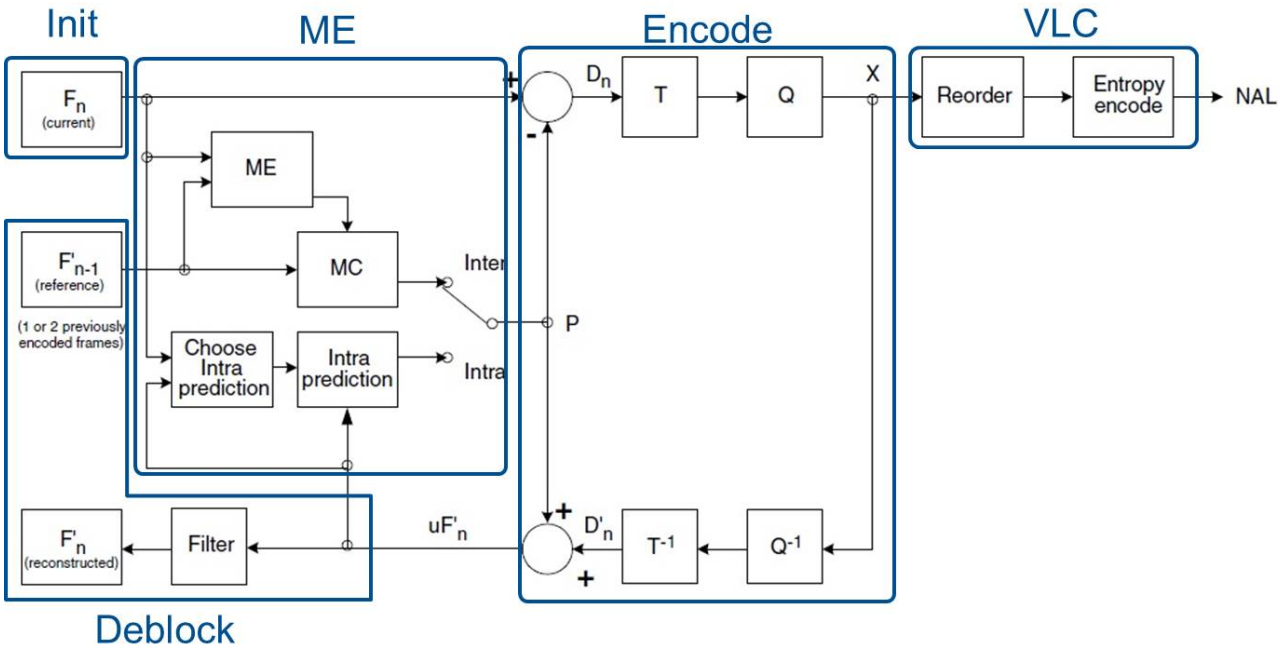

Figure 2-2. Division of the H.264 encoder functions in processes.

2.2.3 Ray Tracing

Providing a high degree of realism, ray tracing is expected to be implemented as a real-time rendering algorithm in the next generation of embedded many-tile systems. Ray tracing applications naturally consist of three logic parts. The first part of the application comprises the generation of the rays. The next part is the second and most computational intensive part of the application, the intersection of the rays with the scene, which is to be rendered. Lastly, the calculated values are aggregated and stored to an image file in the third part. We decided to use this partition for the process network specification, as it allows us to neatly separate the intersection part, which we are interested in parallelizing as it has the biggest contribution to the application's total execution time.

The resulting layout of the process network is illustrated in Figure 2-3. In the resulting implementation, the Generator process calculates all rays and sends them to the Intersect processes, which is where the actual path tracing takes place by calling the process's radiance method, which recursively calls itself until either no object was intersected or the maximum depth has been reached.

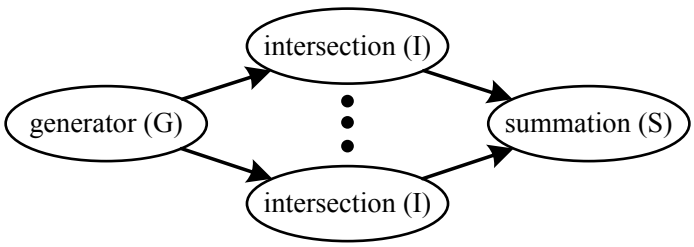

Figure 2-3. DAL specification of the ray-tracing application.

The ray-tracing application has also been used to compare the performance of the proposed design flow with other conventional parallel processing APIs. For instance, in

deliverable 7.3, we have compared the performance of the ray-tracing application when being implemented using the DAL formalism and when implemented using OpenMP. In fact, the throughput of the considered DAL implementation has been almost as fast as a highly optimized OpenMP implementation.

2.3 Other Streaming Applications

In addition, a set of non-multimedia applications has been developed including a recursive array-sorting application, a parallel secure hash algorithm signature application, and a deduplication algorithm application. For brevity, we only summarize the recursive array-sorting algorithm.

2.3.1 Recursive Array-Sorting

The recursive array-sorting algorithm has mainly been developed to test the EPN semantics. In fact, conventional specifications of process networks do not support such recursive applications; however, the proposed semantics of EPNs extends such conventional specifications so that recursive algorithms can be supported.

The considered sorting algorithm is quicksort. The EPN specification is illustrated in Figure 2-4 and Figure 2-5. The top-level process network consists of three processes: Process “src” (“dest”) generates (displays) the input (output) array, and process “sort” sorts the elements in ascending order. As the quicksort algorithm recursively sorts the array, process “sort” can be replaced by a structural description, which divides the array into two smaller arrays that can be individually sorted.

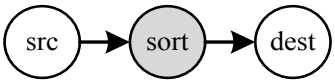

Figure 2-4. Top-level process network of the quicksort algorithm.

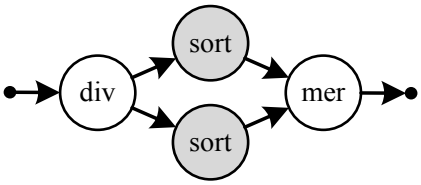

Figure 2-5. Structural description of the process "sort".

The array-sorting algorithm has been successfully applied in various publications including [1 – 5]. For instance, the algorithm has also been used to investigate whenever the EURETILE design flow allows to efficiently exploit the parallelism of a real-world application. The results of this investigation are reported in deliverable 8.3.

2.4 Multi-Application Benchmarks

To demonstrate the expected dynamism in future embedded systems and to illustrate how this dynamism can be described by the DAL formalism, an advanced picture-in-picture (PiP) software for embedded video processing systems is described.

2.4.1 Picture-In-Picture (PiP) Video Decoder

In order to evaluate the multi-application mapping algorithms proposed in [1, 3, 6], we have developed a PiP video decoder software. The benchmark has been developed with

DALipse [2] that allows to visually specify DAL applications and their interactions as a finite state machine.

Figure 2-6 shows a screenshot of DALipse with the finite state machine (FSM) of the considered PiP software. The software is composed of eight scenarios and three different video decoder applications. The *HD* application processes high-definition, the *SD* application standard-definition, and the *VCD* application low-resolution video data. The software has two major execution modes, namely watching high-definition (scenario *HD*) or standard-definition videos (scenario *SD*). In addition, the user might want to pause the video or watch a preview of another video by activating the PiP mode (i.e., starting the *VCD* application). Due to resource restrictions, the user is only able to activate the PiP mode when the *SD* application is running or paused, or the *HD* application is paused.

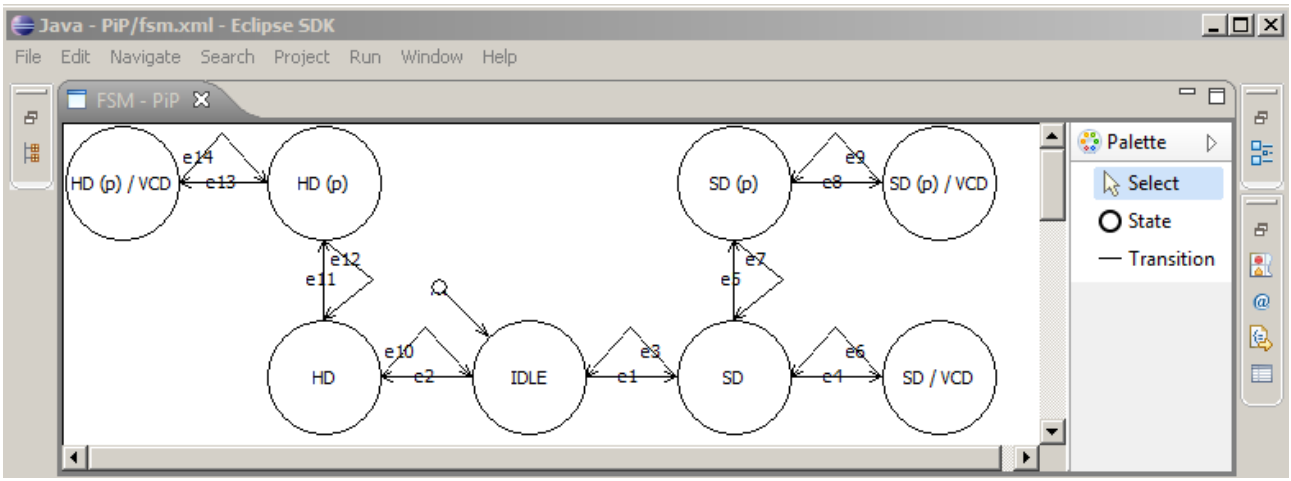

Figure 2-6. Finite state machine of the PiP software. Paused applications are indicated by a (p)

2.5 Conclusion

From 2011 to 2014, in the framework of work package 7, various benchmarks for embedded systems have been developed and successfully applied to evaluate the concepts proposed by the EURETILE project. Besides a set of highly parallel digital signal processing kernels, various multimedia applications have been proposed and used to design complex multi-application benchmarks. In particular, a multi-stage video processing application, a H.264 codec pair, a ray-tracing application, and a distributed sorting application have been implemented using the DAL formalism originally proposed in work package 3. Finally, the proposed benchmarks have been integrated into the DAL framework and are available online for download at http://www.dal.ethz.ch.

3. DPSNN-STDP: large scale plastic spiking neural nets

This chapter reports about the final delivery of the DPSNN-STDP mini-application benchmark for both the DAL and MPI environment, about its improvements during the last project period (Jan-Sep 2014), and about the activities preparing the future exploitation (starting from Oct 2014) in the framework of the CORTICONIC FET project.

DPSNN-STDP implements a Distributed simulator of Polychronous Spiking Neural Network (DPSNN) with Spiking Time Dependent synaptic Plasticity (STDP), and we proved its ability to simulate systems composed of billions of synapses, and its good

strong and weak scaling properties. It is coded as a network of many C++ processes and it designed for a simple porting from the standard MPI distributed programming environment toward the DAL environment.

The reader is referred to [11] for a complete introduction to the DPSNN-STDP mini-application benchmark and its scaling, while the previous deliverable D7.3 (see [9]) discusses the motivations for inclusion of the benchmark in the EURETILE suite. For simplicity of reading, we inserted hereafter a short Section 3.1 that summarizes its main features.

During 2014, a few additional optimization have been added to the MPI reference, as described by Section 3.2. Then, such an optimized code has been ported from MPI to DAL programming environment, for execution on the EURETILE hardware platform, with the support of the DNA-OS operating system (3.3).
During the last project period, we also dedicated some effort to prepare the exploitation of the DPSNN-STDP application in the framework of the CORTICONIC FET FP7 project[1], as described by Section 3.4.

3.1 Reference code (MPI version)

This section summarizes the status of the reference version of the DPSNN-STDP application benchmark (MPI plus C++) (see [11],[9] for a detailed description of its key features).

The global neuro-synaptic network is divided into clusters of neurons and their set of incoming synapses. Each cluster is a node in the network of C++ processes, and it is equipped with a message passing interface. The data structure that describes the incoming synapse includes the information about the total transmission delay introduced by the axonal arborization that reaches it. The list of local synapses is further divided in sets according to the value of the axo-synaptic delay. Each C++ process describes and simulates a cluster of neurons and incoming synapses. The messages travelling between processes are sets of “axonal spikes” (i.e. they carry info about the identity of neurons that spiked and the original emission time of each spike), but do NOT carry info about the set of target neurons. Axonal spikes are sent only toward those C++ processes where at least a target synapse exists for the active set of axons. The knowledge of the original emission time of each spike and of the transmission delay introduced by each synapse allows for the management of synaptic STDP (Spike Timing Dependent Plasticity) (Song, 2000, see [14]), which produces effects of Long Term Potentiation/Depression (LTP/LTD) of the synapses. The simulator uses a mixed event-driven and time-driven integration scheme, someway inspired by [17, 18]. The first implementation of the DPSNN-STDP code implements the Izhikevich neural model described by [15, 16]. Such neural model has been also adopted for their studies of architectures dedicate to large scale neural simulations by [24, 25, 26, 27, 28]. General purpose simulators like [19, 20, 21, 22, 23] are designed to be more flexible, e.g. for what concerns the selection of neural and synaptic models, but their purpose is the detailed simulation of different biological features, while we focus on the study of software and hardware features with a focus on efficient simulation of large scale neural networks, on future HPC and embedded systems (e.g. for robotic applications). For those smaller configurations and simple neural topologies that

[1] The CORTICONIC project is funded through the FET FP7 Grant Agreement no. 600806.

can be run on a single computing node, the DPSNN-STDP distributed code can replicate the results produced by the public access sequential reference code (Izhikevich, 2006, see [13]). During 2013, we performed a first qualitative and quantitative evaluation of the DPSNN performances and its scaling behavior using a standard MPI environment running the application benchmark on the QUonG platform, reported in [11]. QUonG includes sixteen dual socket quad core servers, interconnected both through a 40 Gb/s commodity network, as well as through the APENet+ interconnection system. We used a maximum of 128 physical cores running 2.4 GHz.[2]. Each physical core supported two simultaneous threads.

During these measures, for each neural network size, we checked that the list of spiking neurons and their timings were identical for all run performed using a variable number of software processes and/or physical cores.

In a first set of configurations used for the scaling measures, each neurons projected 200 forward synapses. Neurons were grouped in "columns", each column composed of 1000 neurons (80% excitatory, 20% inhibitory).

Neural columns were organized in bidimensional grids, and each excitatory neuron projected part of its synapses (76%) toward neurons inside the same column, and the remainder was distributed among first, second and third neighboring columns, with decreasing proportions.

The time step of the simulation was set to 1 ms, but the update of the neural membrane potential was performed using a time step of 0.5 ms.

In a second set of measures, performed in 2014, (see section 3.2), we also varied the number of synapses projected by each neuron (from 100 to 10000 synapses per neuron) and the number of neurons per column (from 1000 to 12800 neurons per column).

Table 1. The table reports a subset of measures for significative configuration. The problem sizes varies from 200 K synapses to 6.6 billion synapses. Each neural network size (a column in the table) is distributed using a varying number of MPI processes, and run on a varying number of physical computational resources. The simulation of a given network size produces an identical spiking and plasticity behavior (e.g. firing activity) over 2000 ms of simulated activity, for all distributions among software processes and/or hardware cores.

Total synapses	200 K	800 K	3.2 M	12.8 M	51.2 M	204.8 M	819.2 M	3.2 G	6.6 G
Total neurons	1 K	4 K	16 K	64 K	256 K	1024 K	4.096 M	16.4 M	32.8 M
Grid of neural columns	1 x 1	2 x 2	4 x 4	8 x 8	16 x16	32 x 32	64 x 64	128x128	256x128
Mean firing rate (Hz)	27	24	26	23	22	23	20	22	19
Used cores[3] (min-max)	1-8	1-32	1-128	1-128	1-128	1-128	4-128	64-128	64-128
MPI processes	1-8	1-32	1-128	1–256	1-256	1-256	4-256	64-256	128
Execution time[4] (execution sec / simulated sec)	0.15	0.4	1.80	3.05	6.85	20.0	59	211	386
Normalized execution time: execution time / (firing rate × total syn × simulated second)	**2.73×10^{-8}**	**5.36×10^{-9}**	**2.41×10^{-8}**	**4.22×10^{-9}**	**6.0×10^{-9}**	**4.22×10^{-9}**	**3.61×10^{-9}**	**2.94×10^{-9}**	**3.07×10^{-9}**

[2]Each server is a 1U SuperMicro X8DTG-D. Each node in the cluster is a dual socket. Each socket hosts one quad-core Intel(R) Xeon(R) CPU E5620 (max clock @ 2.40GHz). On each core HyperThreading is enabled (two threads per core). Each node is equipped with a Mellanox InfiniBand board, the MT26428 [ConnectX VPI PCIe 2.0 5GT/s - IB QDR (40Gb/s data rate)]. 16 nodes are connected using a Mellanox Switch.

[3] Each cores @2.4 Ghz part of a quad-core Intel(R) Xeon(R) E5620.

[4] Using the "max" number of cores reported in this table

As discussed in detail in [11,9], the code demonstrated to be fast and scalable: 10 wall clock seconds are required to simulate one second of activity and plasticity (per Hertz of average firing rate) of a network composed by 3.2 Giga synapses running on 128 hardware cores clocked @ 2.4 GHz.

The time needed for the computation needed for each synaptic event (including the management of synaptic plasticity), was about $3x10^{-9}$ elapsed second / synaptic event.

3.2 Improvements during the final period (Jan-Sep 2014)

3.2.1 Increasing the number of synapses projected by each neuron

The execution time of the DPSNN simulator is proportional to a) to the total number of simulated synapses, b) to the firing rate and c) to the physical time to be simulated. During the past years of activity we reported about strong and weak scaling features, using a constant number of synapses projected by each neuron (M=200 in the previous table). Even if the community of computational neuroscientist considers a few hundreds of synapses per neuron adequate for several leading research topics (see for example, Merolla, 2014 [29]), it is necessary to increase M up to several thousands of synapses per neuron when moving toward more detailed biological representations of the human cortex (see, for example Modha, 2011 [26], Izhikevich[28]). Therefore, during 2014, we structured the synaptic database to easily accommodate a higher number of projected synapses per neuron, and verified the efficiency of the solution on the desired range. Table 3-2 reports about the relative execution time per synaptic update, varying the number of project synapses.
A notable feature: the execution time per simulated synapse improves when the number of synapses projected by each neuron increases toward the figure required for simulating the human cortex.

Table 3-2 - The number of synapses projected by each neuron has been varied (from M=100 to M=10000). A notable feature: the execution time per synapse improves when the number of synapses projected by each neuron increases toward biologically realistic figures.

M: Synapses projected by each neuron	**100**	**200**	**1 000**	**1 000**	**10 000**
Relative execution time	**1.00**	**0.77**	**0.90**	**0.74**	**0.64**
Total synapses	104 M	104 M	164 M	164 M	128 M
Total neurons	1.04 M	524 K	164 K	164 K	12.8 K
Neurons per cortical module	1 024	1 024	10 240	1 280	12 800
Cortical modules	1 024	512	16	128	1
Grid of cortical modules	32 x 32	32 x 16	4 x 4	16 x 8	1 x 1
Mean firing rate (Hz)	12.78	10.86	10.21	9.34	9.95
Used cores (min-max)	32	32	32	32	32

3.2.2 Additional optimization: storage of future events and partition of delays

During 2014, a strong improvement in terms of memory usage (and also some gain in execution efficiency) has been obtained by creating delay queues at the interface between incoming spike messages and the set of backward synapses managed by each process.

When a neuron spikes, a message containing information about the spike (original emission time and identification number of the firing neuron) is sent to all the processes that contain target neurons for the set of active axons. Once the axonal spike messages reaches a target processes, it must be forwarded to a set of target neurons through backward synapses. Each synapse introduces an individual temporal delay, thus the set of target neurons to be reached by a single spike can be partitioned in subsets of neurons sharing the same synaptic delay.
A current must be injected in the target neuron, proportional to the strength of the interconnecting synapse, at a moment that can be computed as the sum between the original emission time of the spike and the specific synaptic delay.
In the previous implementation (2013 version) of the neuronal simulator, all the spikes were immediately delivered to the set of target neurons and there stored as events to be managed in the future. Indeed, the delays were inserted using circular buffers (i.e. the "synaptic spike schedulers") associated to each target neuron.
We partitioned the data base of incoming synapses according to their delay. Then we added delay buffers for incoming axonal spikes at the incoming message interface. This permitted to eliminate the memory needed to store future events on each target neuron. The amount of saved storage is substantial. For example, suppose there are M synapses per neuron, and assume that they are homogeneously distributed over D set according to temporal delays. This strategy will reduce the memory consumption associated to the storage of future events by a factor M/D. As a result, synaptic currents will be injected into the target neuron at the appropriate simulation time.
The improvement, mainly motivated by this potential gain of a factor M/D in memory associated to the storage of synaptic events, produced also an improvement of execution times (e.g. a gain of ~7% has been obtained for N=1048576, M=100, P=16), As a collateral effect, the suppression of the functions associated to synaptic spike schedulers simplified the code.

3.2.3 Random synapses and external inputs for deterministic behaviour

A requirement we fixed for the distributed parallel DPSNN simulator is that it should be possible to run it in a mode where the results are reproducible when running the same problem size on a varying number of processes. This modality is useful to simplify the validation of the distributed implementation and to perform more accurate strong and weak scaling measures.
During the initialization phase, each neuron generates a set of forward synapses that connect it to a subset of others neurons in the network. The subset of neurons toward which it connects should be generated using a combination of deterministic and random criteria.
For example, there could be an underlying regular bidimensional grid of neural columns, and probabilities of interconnection between each column and the others. Also, during the simulation phase, an external stimulus should be provided to the neural network (e.g, simulating a thalamic input). A combination of deterministic and probabilistic criteria can be used to identify the neurons that have to be fed with the external input, time by time.
The standard rand() function generates a sequence of random numbers on each process. If, once fixed the size of the problem, we used the standard rand() function for a different number of software processes , we would produce different simulation results, e.g. for one, two or 4 processes. In order to satisfy the requirement of application reproducibility when scaling on a different number of processes, a specific implementation of the random

generation has been designed, which wraps the C++ random number generator rand() using adequate seeds.
In addition to this, a distribution of probability has been introduced for the random thalamic input. For a fixed thalamic input frequency, a distribution of probability is generated at each simulation step in order to have a more flexible input generator. In future development, it can be interfaced with an appropriate number generator (uniform, exponential, etc.) in order to have the desired probability distribution function.

3.2.4 Analysis of the Izhikevich model dynamic

The configuration of the DPSNN simulator used as EURETILE benchmark implements neurons based on the Izhikevich model [15,16], whose equations are reported hereafter:

$$\begin{cases} \boldsymbol{if}\ \boldsymbol{v}(\boldsymbol{t}) < v_{peak} \quad then \quad \begin{cases} \dot{\boldsymbol{v}} = \mathbf{0.04}\,\boldsymbol{v}^{\mathbf{2}} + 5\boldsymbol{v} + \mathbf{140} - \boldsymbol{u} + \boldsymbol{I} \\ \dot{\boldsymbol{u}} = \boldsymbol{a}(\boldsymbol{bv} - \boldsymbol{c}) \end{cases} \\ \boldsymbol{if}\ \boldsymbol{v}(\boldsymbol{t}) \geq v_{peak} \quad then \quad \begin{cases} \boldsymbol{v}(\boldsymbol{t} + \Delta\boldsymbol{t}) = \boldsymbol{c} \\ \boldsymbol{u}(\boldsymbol{t} + \Delta\boldsymbol{t}) = \boldsymbol{u}(\boldsymbol{t}) + \boldsymbol{d} \end{cases} \end{cases}$$

where:

- $\mathrm{v}(\mathrm{t})$ represents the neural membrane potential. We say that when $\boldsymbol{v}$ reaches v_{peak} a "neural spike" happens;
- I(t) is the potential change generated by the sum of all synapses incoming to the neuron. Incoming currents are present if spikes arrive form presynaptic neurons;
- u(t) represents a membrane recovery variable;
- a, b, c, d are four parameters, constant for each neuron kind, by varying them the same equation models several kind of known neural types;
- After a spike, the membrane potential and the recovery variable are reset using the c and d reset constants.

During 2014, the dynamic of the Izhikevich model has been studied in detail, with the help of some Scilab simulations, with the purpose of improving the stability of the simulations for a varying number of projected synapses. As a result of this analysis, a small correction to the standard implementation of the neuron numerical dynamic has been done that proved to increase the stability of the simulations: when the membrane potential reaches the v_{peak} value (i.e. v>=30 mV and the neuron fires), the recovery variable u is not updated using the new value of v, but it's updated using the value of v clipped to 30 mV. Then, both u and v are reset to the appropriate values during the after-spike dynamic, as in the usual Izhikevich dynamic.
We observed that clipping the membrane voltage as soon as the neuron fires, produces an enhancement of the system stability. In particular, the behavior of the u variable seems to be more realistic: using the Izhikevich model with this modification there are no more huge extra picks in correspondence of every neuronal spike, as instead observed in the same model without the correction, and the observed variance of the instantaneous firing rates of the networks reduces significantly.

3.2.5 Quantitative evaluation of the inter-processes message size

Both in initialization and in simulation, a mechanism of message passing is used for communication among cluster of neurons that populate the network. DPSNN adopts a mixed event-driven, time-driven approach. In a system composed by N neurons, each one projecting M synapses, the simulation cost is dominated by synaptic events, that are managed using an event-driven mechanism. Instead, the simulation of neural dynamics is performed using a time-driven integration scheme. At every time iteration, each cluster of neurons send a set of axonal spikes to all the neuron clusters (i.e. “processes”) in the network where a target synapse exists, where the events are generated This delivery of spiking messages can be split in two steps, with communications directed toward subsets of decreasing sizes.
During the first step, single word messages (spike counters) are sent to the subset of potentially connected target processes. On each pair of source-target process subset, the individual spike counter informs about the actual payload (i.e. axonal spikes) that will have to be delivered, or about the absence of spikes to be transmitted between the pair.
The second step uses the spiking counter info to establish a communication channel only between pairs of processes that actually need to transfer an axonal spikes payload during the current simulation time iteration.
A quantitative evaluation of this inter-process communication has been done, measuring the size of the messages exchanged in both steps by each process. The values are collected in an output file containing a set of statistical measurements. In detail, the MessageDimSize reports the size in byte of the data exchange during the first step, while the MessagePayloadSize reports the corresponding values in the second step (the actual payload). For each measurement, several values are reported (min, max, mean, variance, etc.) useful for statistical purposes.
These measures have been used during the implementation of the message passing mechanism through the APENet+ communication interface, in order to get performance evaluations of the custom network controller, compared with the Infiniband implementation. The results of these analysis are reported in the deliverable D6.4 (HW developments).

3.2.6 Scripts for analysis of performances and simulation results

During 2014, some bash and Scilab scripts have been developed in order to simplify the analysis of the results obtained with different run of the DPSNN simulator.
Among the bash scripts, one of the more useful is the *average* script, that is used to calculate the average of a specific measure, for example the time spent over a certain routine, over the set of active processes. The script uses the values written in statistical files by each process.
Among the Scilab scripts, we also developed some useful scripts used to produce the rastergram and the instantaneous spiking rate, starting from the file that collects the spikes produced by the network; a script used to plot the dynamic of an Izhikevitch neuron in his main components: I (input current), V (membrane potential), U (recovery variable); a script for the frequency analysis and the power spectrum generation.
All these scripts are available for download together with the source code of the DPSNN application.

3.3 Porting DPSNN to DAL

During 2014, the MPI version of the DPSNN-STDP application described in Section 3.1, and the improvements described in Section 3.2, have been ported to the DAL programming environment for execution on the EURETILE hardware platform, with the support of the DNA-OS operating system.
In order to maintain all the functionalities of the MPI code and almost exactly the same files, the parameters that in the MPI version are defined at compile time through the Makefile, in the DAL environment are passed to the application using some *#define* statements.
D7.1 (report of 2011 activities) and D7.2 (2012 activities) described the basic methodology adopted for the MPI->DAL porting.

In addition, during 2014, some modification to the scripting system used to run the DAL version of DPSNN has been done. In detail, a new script has been created, for the automatic generation of the xml files used to describe the network. The user can compile a DAL version for the desired number of DAL processes invoking the script "*makeMappingScript.sh*". The script generates the two files required by the DAL environment, specific for each application:

- the process network file, used to describe the application at an abstract level;
- the mapping file, that defines where and how the components of the application are executed on a distributed hardware platform.

Moreover, an additional modification has been done to the "*dalrunDPSNNf*" script used to launch the DAL version of DPSNN. The script, invoked specifying the number of processes in which the user aim to divide the neural network, provides to automatically call the "*makeMappingScript.sh*", which operates as described above. As a result, to run the DPSNN application on a specific number of processes, is enough to call the following statement:

```
>  source dalrunDPSNNf N
```

where N is the number of processes (1, 2, 4, 8… 2^n).
In conclusion, the delivered DAL version exampleDPSNNf application (euretile svn revision 431, 8 August 2014, available to all consortium), is aligned, in terms of code and functionalities, with the DPSNN application MPI version (ape svn revision 822, 11 June 2014). Both versions produce the same results when run with the same configuration parameters. The version used for final experimements on the integrated EURETILE software tool-chain on QUonG is the euretile svn revision 441, 17 oct 2014, aligned with MPI revision 850)

3.4 Preparing the exploitation in FET CORTICONIC project

CORTICONIC is an ICT-FET project, funded through the ICT-2011.9.11 - FET Proactive Neuro-Bio-Inspired Systems (NBIS), in the framework of FP7 Work Program.
CORTICONIC stands for Computations and Organization of Retes Through the Interaction of Computational, Optical and Neurophysiological Investigations of the Cerebral cortex. The aim of this project is that to identify and understand the computational principles of the cerebral cortex through the study of the electrical activity generated by the cortical network, using both experimental data from in-vivo and in-vitro measurements, but also a theoretical and computational approach based on a neural network simulation.

The techniques adopted for the development of the parallel and distributed neural network simulator DPSNN developed in EURETILE could be used as a baseline to build a simulator able to satisfy the requirement of the CORTICONIC project, starting from October 2014.
During the last few months, some effort was dedicated to the preparation of this immediate exploitation. In particular, a new kind of neuron (LIFCA), required by the CORTICONIC project, has been introduced. Moreover, the input currents feeding this neuron from the external (analogues to the thalamic input for the Izhikevich neuron) was modeled with more detail, according to the project requirements.
Several other improvements to the DPSNN simulator will be necessary in order to provide a full support to the CORTICONIC needs. As a natural follow up, the future enhancements to the DPSNN simulator will be desirably implemented in the framework of the CORTICONIC project.

3.4.1 LIFCA neuron dynamic

The additional neuron model implemented in the DPSNN simulator is the **L**eaky **I**ntegrate and **F**ire neuron with spike frequency **C**urrent **A**daption (LIFCA). The neuron dynamic equations are:

$$\begin{cases} \boldsymbol{if}\ \boldsymbol{v(t)} < \boldsymbol{v_\vartheta} \quad then \quad \begin{cases} \boldsymbol{\tau_v \dot{v} = -v + R\,(I - c)} \\ \boldsymbol{\tau_c \dot{c} = -c} \end{cases} \\ \boldsymbol{if}\ \boldsymbol{v(t)} \geq \boldsymbol{v_\vartheta} \quad then \quad \begin{cases} \boldsymbol{v(t + \Delta t) = v_{reset}} \\ \boldsymbol{c(t + \Delta t) = c(t) + \alpha_c} \end{cases} \end{cases}$$

where:

- $\boldsymbol{v(t)}$ is the membrane potential;
- $\boldsymbol{c(t)}$ is the adaption variable due to Ca^+ currents
- $\boldsymbol{I}$ are the incoming currents
- $\boldsymbol{R}$ is membrane resistance
- $\boldsymbol{v_\vartheta}$ is the voltage threshold
- $\boldsymbol{v_{reset}}$ is the reset membrane potential
- $\boldsymbol{\tau_v}$ is the decay time constant of $\boldsymbol{v}$
- $\boldsymbol{\tau_c}$ is the decay time constant of $\boldsymbol{c}$
- $\boldsymbol{\alpha_c}$ is the post-spike Ca^+ concentration increment

When the membrane potential reach the voltage threshold value $\boldsymbol{v_\vartheta}$, the neuron fires. After the spike, the membrane potential is reset to the reset membrane potential value $\boldsymbol{v_{reset}}$ and the adaption variable is added with the post-spike Ca^+ concentration increment $\boldsymbol{\alpha_c}$.

An example of the dynamic of the LIFCA neuron, as implemented in the DPSNN simulator, is depicted in Figure 2-1. There, the neuron emits two spikes, the first one at about 749 ms, and the second one at 819 ms. In correspondence of the two spikes, the adaption variable is increased of the value corresponding to the post-spike Ca^+ concentration increment. The current injected at each step is reported in the picture (lower curve). It is

the sum of the external "thalamic" currents and the current internal to the network, generated by pre-synaptic spiking neurons.

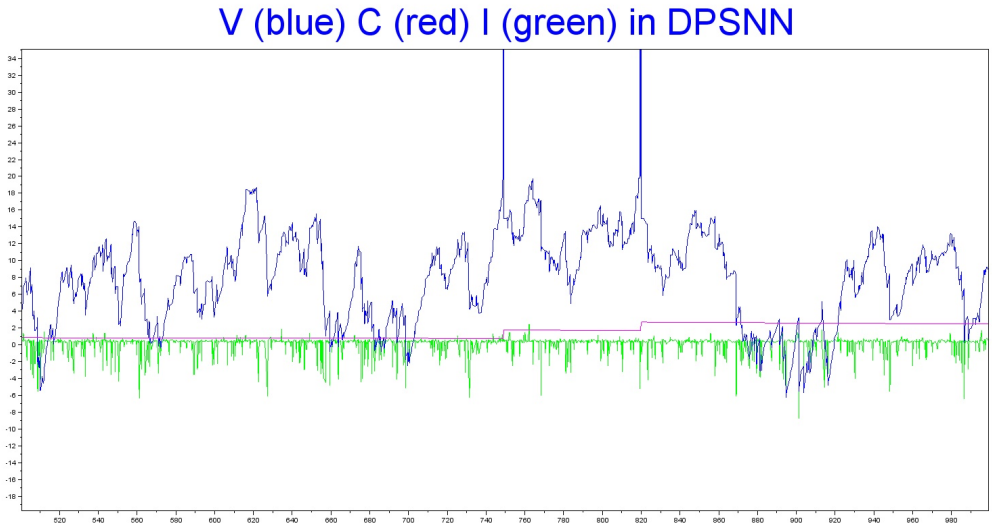

Figure 3-1 – LIFCA neuron dynamic. The membrane potential is depicted in blue, the adaption variable in red, the input currents in green.

3.4.2 LIFCA neuron network

While the simulation with the Izhikevich model has been already tested up to billion of synapses, as described by Section3.1 [11], the LIFCA neuron network, implemented during the last months in DPSNN is still under test at the moment (August 2014). For this preliminary test we are distributing among a variable number of software processes a network of fixed dimensions: 2500 neurons (2000 excitatory neurons, plus 500 inhibitory neurons) each one projecting 1000 synapses,
The two population of neurons, excitatory and inhibitory neurons, are interconnected between them, using the following connection probability

exc --> exc with 80% of probability connection
exc --> inh with 20% of probability connection
inh --> exc with 80% of probability connection
inh --> inh with 20% of probability connection

This first toy-level implementation has been mainly used to study the correctness of the LIFCA dynamic coded in the DPSNN. In future activities inside the CORTICONIC project, the network will be generalized, in order to allocate a customizable number of neurons populations with parametric probability connections.
In order to provide a complete and efficient simulator useful to the CORTICONIC community, several other generalization will have to be added to the DPSNN application, including a more flexible distribution of: 1-synaptic delays, 2- initial synaptic weights, 3- external input current, 4- connectivity among neural populations.

3.4.3 Sub millisecond resolution and refractory period

In the implementation required by CORTICONIC, arrival times of external input currents are expected to be generated following a Poisson distribution. The desired precision for

arrival times is well below the millisecond. In order to maintain this kind of precision, we devised a strategy that takes benefit of the existence of a refractory post-spiking period simulate at sub millisecond precision using an improvement to the DPSNN mixed event-driven, time driven scheme, maintaining an inter-process message passing phase carried every ms. At each “time-driven” simulation step (1 ms) each neuron is checked for the appropriate number of external inputs (with Poisson distribution) that arrive in the millisecond. Every time an input is encountered, the dynamic on that input is completed (update of v(t) and c(t) variables, with check on spike condition and eventual after-spike dynamic). This means that if the external input frequency is set to 3 Hz, the neuron dynamic on external inputs is executed a mean value of three times per neuron per millisecond, maintaining a correct sub millisecond temporal behaviour. In addition to the external input current, each neuron could receive also the current from firing pre-synaptic neurons, but the existence of a refractory period guarantees that each pre-synaptic neuron will not fire more than once per ms. These currents are associated with a random arrival time value internal to the current millisecond and are inserted in the dynamic of the neuron with the same modality used for external inputs.

3.5 Conclusion

At the end of the EURETILE project, the DPSNN application benchmark is available in both the MPI and the DAL version. The benchmarks have been used to measure performances of the EURETILE platforms and to drive the development of future parallel/distributed computing systems dedicated to the simulation of plastic spiking networks. The DPSNN code will find immediate exploitation in the framework of the CORTICONIC project: the parallel and distributed neural network simulator developed within EURETILE will be the kernel around which build a more complex simulator, able to satisfy the requirement of the CORTICONIC community.

4. LQCD kernels

LQCD is one of the first applications developed for parallel and distributed computers, and has been one of the scientific drivers of the activities of the APE lab of INFN since 1984. Indeed, LQCD played a key role for the activities of APE lab, setting main requirements for the co-design of software and the hardware of several generations of massive parallel computers designed by INFN (APE [30], APE100[31], APEmille[32], APENext[33]) and its software tool-chain [34,35,36] along with first neural networks models [37], climatology [38], lattice-boltzmann simulations [39], multidimensional matrix transposition for FFT based applications (e.g. seismic migration and synthetic array radars) [40].
One deliverable of the SHAPES project [41], predecessor of the EURETILE project, has been the coding. In cooperation with ETHZ, of the LQCD computational kernel as a homogeneous bidimensional Khan network of processes, using a data partitioning scheme that divides the 4-dimentional spatiotemporal LQCD lattice into a bi-dimensional grid of processes that communicates with first neighbor processes.
Previous deliverables (D7.1 - about 2011 application activities, and D7.2 – about 2012 developments) described the LQCD kernel coded as a DAL network of processes.
During 2014, no additional activity was required about LQCD.

References

[1] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele. Scenario-Based Design Flow for Mapping Streaming Applications onto On-Chip Many-Core Systems. Proc. Int'l Conf. on Compilers Architecture and Synthesis for Embedded Systems (CASES), pp. 71–80, 2012.

[2] L. Schor, D. Rai, H. Yang, I. Bacivarov, and L. Thiele. Reliable and Efficient Execution of Multiple Streaming Applications on Intels SCC Processor. Proc. Workshop on Runtime and Operating Systems for the Many-core Era (ROME), Aachen, Germany, August 2013.

[3] L. Schor, I. Bacivarov, L. G. Murillo, P. S. Paolucci, F. Rousseau, A. El Antably, R. Buecs, N. Fournel, R. Leupers, D. Rai, L. Thiele, L. Tosoratto, P. Vicini, and J. Weinstock. EURETILE Design Flow: Dynamic and Fault Tolerant Mapping of Multiple Applications onto Many-Tile Systems. Proc. IEEE Int'l Symposium on Parallel and Distributed Processing with Applications Article (ISPA), Milan, Italy, Aug. 2014.

[4] L. Schor, H. Yang, I. Bacivarov, and L. Thiele. Expandable Process Networks to Efficiently Specify and Explore Task, Data, and Pipeline Parallelism. Proc. Int'l Conf. on Compilers Architecture and Synthesis for Embedded Systems (CASES), pages 1–10, Montreal, Canada, Oct 2013.

[5] L. Schor, I. Bacivarov, H. Yang, and L. Thiele. AdaPNet: Adapting Process Networks in Response to Resource Variations. Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), New Delhi, India, Oct. 2014.

[6] S.-H. Kang, H. Yang, L. Schor, I. Bacivarov, S. Ha, and L. Thiele. Multi-Objective Mapping Optimization via Problem Decomposition for Many-Core System. Proc. IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia), Tampere, Finland, pp. 28-37, 2012.

[7] L. Schor, A. Tretter, T. Scherer, and L. Thiele. Exploiting the Parallelism of Heterogeneous Systems using Dataflow Graphs on Top of OpenCL. Proc. IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), Montreal, Canada, p. 41-50, 2013.

[8] S. Kwon, Y. Kim, W.-C. Jeun, S. Ha, and Y. Paek. A Retargetable Parallel-Programming Framework for MPSoC. ACM Trans. Design Autom. Electr. Syst., 13, 2008.

[9] P.S. Paolucci et al. EURETILE D7.3 – Dynamic DAL benchmark coding, measurements of MPI version of DPSNN-STDP (distributed plastic spiking neural net) and improvements to other DAL codes. arXiv:1408.4587 [cs.DC]

[10] Paolucci, P.S., Bacivarov, I., Goossens, G., Leupers, R., Rousseau, F., Schumacher, C., Thiele, L., Vicini, P., "EURETILE 2010-2012 summary: first three years of activity of the European Reference Tiled Experiment.", arXiv:1305.1459 [cs.DC], (2013), http://arxiv.org/abs/1305.1459

[11] P.S. Paolucci et al. Distributed simulation of polychronous and plastic spiking neural networks: strong and weak scaling of a representative mini-application benchmark executed on a small-scale commodity cluster. arXiv:1310.8478 [cs.DC], (2013)

[12] http://www.corticonic.org

[13] Izhikevich, E. M. Polychronization: Computation with Spikes. Neural Computation, 18, 245-282 (2006).

[14] S. Song, K.D. Miller, L.F. Abbott. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. Nature Neuroscience 3, 919-926, (2000)

[15] Izhikevich, E. M. Simple Model of Spiking Neurons. IEEE Transactions on Neural Networks, Vol. 14, No. 6, November 2003

[16] Izhikevich, E. M. Which Model to use for cortical spiking neurons? IEEE Transaction on Neural Networks, 15, no. 5 1063-1070 (2004) .

[17] P. Del Giudice, M. Mattia. Efficient Event-Driven Simulation of Large Networks of Spiking Neurons and Dynamical Synapses. Neural Computation, 12, 2305-2329 (2000).

[18] A. Morrison, C. Mehring, T. Geisel, A. Aertsen, M. Diesmann. Advancing the Boundaries of High-Connectivity Network Simulation with Distributed Computing. Neural Computation, 17, 1776-1801 (2005).

[19] N.T. Carnevale, M.L Hines. The NEURON Book. Cambridge University Press, (2006)

[20] N. T. Carnevale, M. L. Hines. NEURON for Empirically-Based Simulations of Neurons and Networks of Neurons. Available online at: http://www.neuron.yale.edu/neuron/ (retrieved October 2013).
[21] J.M Bower, D. Beeman. The Book of GENESIS. Exploring realistic neural models with the GEneral NEural SImulation System (1988).
[22] GENESIS: GEneral NEural SImulation System. Available online at thhp://www.genesis-sim.org (retrieved Ocotber 2013)
[23] Gewaltig M-O & Diesmann M. (2007) NEST (Neural Simulation Tool) Scholarpedia 2(4):1430.
[24] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, A. D. Brown. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, vol. PP, no.99, (2012). doi: 10.1109/TC.2012.142.
[25] Xin Jun, Steve B. Furber, John V. Woods. Efficient Modelling of Spiking Neural Networks on a Scalable Chip Multi-processor. *Int. Joint Conf. on Neural Networks 2008 (IJCNN 2008), 2812-2819 (2008)* .
[26] Modha, S. D., & al., e. (2011). Cognitive Computing. *Communications of the ACM , 54* (08) 62-71.
[27] Jayram M. Nageswaran et al. (n.d.). Efficient Simulation of Large-Scale Spiking Neural Netwoks Using CUDA Graphics Processors. *Int. Joint Conf. on Neural Networks 2009 (IJCNN 2009), 2145-2152 (2009).*
[28] E. M. Izhikevich, G. M. Edelman. Large-scale model of mammalian thalamocortical systems. PNAS March 4, 2008 vol. 105 no. 93593-3598.
[29] Paul A. Merolla, & al., (2014). A million spiking-neuron on integrated circuit with a scalable communication network and interface*, Science 345, 668 (2014)* (pag. 668-673).
[30] M. Albanese et al. The APE computer: An array processor optimized for lattice gauge theory simulations, Computer Physics Communications, 45, Issues 1–3, (1987) 345-353, ISSN 0010-4655, http://dx.doi.org/10.1016/0010-4655(87)90172-X.
[31] C. Battista et al, The APE-100 Computer: (I) The Architecture. Int. J. High Speed Comp. 05, 637 (1993). DOI: 10.1142/S0129053393000268
[32] F Aglietti et al. An overview of the APEmille parallel computer. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 389, Issues 1–2, 11 April 1997, Pages 56-58, ISSN 0168-9002, http://dx.doi.org/10.1016/S0168-9002(97)00040-5.
[33] F. Belletti et al. Computing for LQCD: apeNEXT. Comput. Sci. Eng. 8, 18 (2006); http://dx.doi.org/10.1109/MCSE.2006.4
[34] S. Cabasino, Pier S. Paolucci, and G. M. Todesco. 1992. Dynamic parsers and evolving grammars.SIGPLAN Not. 27, 11 (1992), 39-48. http://doi.acm.org/10.1145/141018.141037
[35] A. Bartoloni et al, Int. J. Mod. Phys. C 04, 955 (1993). The Software of the APE100 Processor. DOI: 10.1142/S0129183193000732
[36] P.S. Paolucci, R. D'Autilia, G.M. Todesco, S. Cabasino. The TAO Language. (1995). https://sites.google.com/site/pierstanislaopaolucci/tao-language-book.
[37] P.S. Paolucci, N-Body Classical Systems and Neural Networks on a 3D SIMD Massive Parallel Processor: APE100/QUADRICS. Int. J. Mod. Phys. C, 06, 169 (1995).
[38] C. Ronchi, R. Iacono, P.S. Paolucci, The "Cubed Sphere": A New Method for the Solution of Partial Differential Equations in Spherical Geometry, Journal of Computational Physics, 124, 1, 93-114, (1996) http://dx.doi.org/10.1006/jcph.1996.0047.
[39] A. Bartoloni et al. LBE Simulations of Rayleugh-Bénard Convection on the APE100 Parallel Processor. Int. J. Mod. Phys. C 04, 993 (1993). DOI: 10.1142/S012918319300077X
[40] N. Cabibbo and P.S. Paolucci, SIMD Algorithm for Matrix Transposition. Int. J. Mod. Phys. C 06, 183 (1995). DOI: 10.1142/S0129183195000149
[41] Paolucci, P.S.; Jerraya, AA; Leupers, R.; Thiele, L.; Vicini, P., "SHAPES:: a tiled scalable software hardware architecture platform for embedded systems," Hardware/Software Codesign and System Synthesis, 2006. doi: 10.1145/1176254.1176297