Optimizing Network Traffic for Spiking Neural Network Simulations on Densely Interconnected Many-Core Neuromorphic Platforms

Optimizing Network Traffic for Spiking Neural Network Simulations on Densely Interconnected Many-Core Neuromorphic Platforms

Gianvito Urgese, *Student Member, IEEE,* Francesco Barchi, Enrico Macii, *Senior Member, IEEE* and Andrea Acquaviva, *Member, IEEE*

Abstract—In this paper we present a new Partitioning and Placement methodology able to maps Spiking Neural Network on parallel neuromorphic platforms. This methodology improves scalability/reliability of Spiking Neural Network (SNN) simulations on many-core and densely interconnected platforms. SNNs mimic brain activity by emulating spikes sent between neuron populations. Many-core platforms are emerging computing targets that aim to achieve real-time SNN simulations. Neurons are mapped to parallel cores, and spikes are sent in the form of packets over the on-chip and off-chip network. However, the activity of neuron populations is heterogeneous and complex. Thus, achieving an efficient exploitation of platform resources is a challenge that often affects simulation scalability/reliability. To address this challenge, the proposed methodology uses customised SNN to profile the board bottlenecks and implements a SNN partitioning and placement (SNN-PP) algorithm for improving on-chip and off-chip communication efficiency. The cortical microcircuit SNN was simulated and performances of the developed SNN-PP algorithm were compared with performances of standard methods. These comparisons showed significant traffic reduction produced by the new method, that for some configurations reached up to 96X. Results demonstrate that it is possible to consistently reduce packet traffic and improve simulation scalability/reliability with an effective neuron placement.

Index Terms—Neuromorphic Platform, Many-core SoC, Profiling Methodology, Spiking Neural Network, Partitioning and Placement.

1 Introduction

The simulation of Biological Neural Networks (BNN), the structures composing neural tissue, is a promising methodology to gain novel insights into unclear mechanisms underlying brain functions. The high degree of complexity that characterizes the nervous system poses several challenges when dealing with the simulation of its processes using abstract models. BNNs are usually represented during the simulations as Spiking Neural Networks (SNNs) [1], interconnected neuron models which mimic the behaviour of real neurons and which communicate using spikes. Challenges of SNN simulations are related to the computational and communication effort needed to account for the large number of neurons and in particular of synapses, that are the interconnections between them. This effort cannot be tackled using general-purpose computation platforms only. During the last decade, a number of custom hardware (HW) architectures have been developed, aimed at supporting neuron activity with enough parallelism and implementing efficient spike communication by means of densely interconnected networks [2, 3, 4, 5].

The SpiNNaker platform [6] is one of the most advanced neuromoprhic solutions which is also one of the two currently adopted in the Human Brain Project (https://www.humanbrainproject.eu/) to simulate SNNs. SpiNNaker is a globally asynchronous locally synchronous (GALS) application-specific multi-chip many-core architecture used for the execution of real time simulations of SNNs. The system is organized in a two-dimensional toroidal-shaped triangular mesh where SpiNNaker chips represent the processing nodes (Figure 1). Each node contains an on-chip router and 18 ARM968 cores for the parallel execution of a variety of neuronal models and synapses. Neuron models are mapped on the cores of SpiNNaker chips and their spikes are propagated across the network in the form of packets. SpiNNaker is made of general-purpose cores and for this reason it can be potentially used for a wide range of applications requiring intensive communication between parallel computational elements [7].

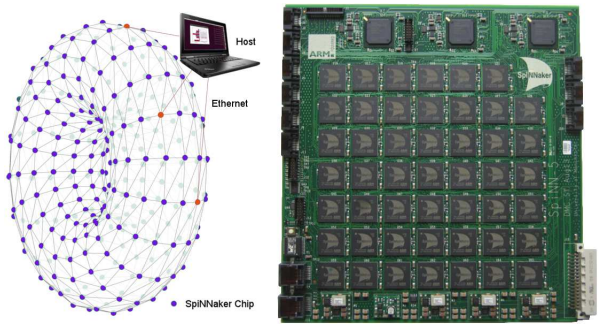

Fig. 1: **SpiNNaker board** with 48 multi-core chips connected in a toroidal-shaped triangular mesh.

Given the complexity of the communication activity in simulated SNNs, a significant challenge is to reduce the risk of unreliable simulation behaviour and failures in the absence of an efficient exploitation of platform architectural

- *G. Urgese, F. Barchi, E. Macii and A. Acquaviva are with the Department of Control and Computer Engineering - DAUIN, Politecnico di Torino, Torino 10129, Italy (e-mail: gianvito.urgese@polito.it).*

Manuscript received January 07, 2016; revised April 25, 2016.

resources. In particular, partitioning and placement of neuron populations into chips and cores heavily impacts the efficiency of on-chip and off-chip communication during the simulation.

In this paper, we describe a partitioning and placement algorithm for SNN simulations over neuromorphic platforms. This algorithm is designed to decrease the traffic of packets over the inter-chip network and to improve simulation reliability and communication efficiency. Thus, the way is paved to more reliable and scalable SNN simulation.

The methodology has been developed across two main phases: A top-down profiling analysis was performed first to detect bottlenecks in the SpiNNaker communication system. Then, a SNN Partitioning and Placement (SNN-PP) algorithm inspired by the profiling results was developed. The SNN-PP algorithm exploits clustering and legalization techniques to achieve a more efficient mapping of SNN simulation components to the platform.

The paper is organized as follows. Section 2 provides an overview of the SpiNNaker SW/HW architecture. Section 3 describes the profiling methodology and highlights the effectiveness of the methods used to study the packet traffic in two path locations. A biological SNN simulation is used to validate the evidences observed. The use of the customised partitioning and mapping algorithm can improve the standard SNN placement procedure by avoiding identified configurations that lead to unreliable behaviour. Section 4 describes SNN-SA-PP, the SNN Partitioning and Placement algorithm based on Spectral Analysis methods. Section 5 reports on the results obtained during the SNN-SA-PP validation process performed on a biological SNN simulation. Section 6 provides the final evaluations.

2 Background

The *Neuromorphic engineering* aims at developing VLSI systems to mimic the neuro-biological networks of the nervous system. A biological neuron collects signals from its predecessors (pre-synaptic neurons) and transmits a spike if the membrane voltage reaches the firing threshold value, otherwise the potential reached will decay over time. This spike is transmitted along a wire called *axon* to the connection with the dendrite of other neurons called post-synaptic. The axon-dendrite contact is called *synapse*. Each synapse is characterized by a specific weight that influences the changes induced by the pre-synaptic spike in the membrane electric potential of post-synaptic neuron. On average, the neuron spiking rate (spike/second) ranges from 10 to 100 Hz. The nervous system networks make the importance of each single neuron relatively low. This is due to its very high level of parallelism and its ability to adapt to unknown environments. Remarkable fault tolerance is provided even after the loss of many neurons.

Neuromorphic SW/HW systems support the simulation of the nervous system. They allow the study of the working mechanisms acting in the brain and to investigate the biological process underlying neural diseases. At the same time, neuromorphic engineers take inspiration from biology to design brain-like systems with brain-specific features. These include extreme parallelism, adaptive responsiveness to unknown environments, fault-tolerance, and very low-power consumption [8].

2.1 Spiking Neural Network Simulation

Spiking Neural Networks (SNN) are neural networks adopted to simulate brain activity in a biologically plausible way. In the SNN simulations neurons and their synapses are modelled as differential equations. These equations are capable to emulate the behaviour observed in biological networks, making possible to describe network dynamics and mechanisms [1]. Two of the most adopted neuron models are the *leaky Integrate and Fire (IF)* [9] and *Izhikevich (IZK)* [10], because their accuracy enables the exploitation of SNN for accurate neuron dynamics observation, exploration, and validation of plausible theories regarding brain functions at an affordable computational time. Moreover, SNN simulations can reproduce the experiments with the same conditions.

Overall, a SNN can be described as a graph where each node, called *Population*, is a homogeneous group of neurons sharing the same model and parameters. Whereas, each edge (*Projection*) is the rule used to generate synaptic connections between the neurons of two *Populations*. Nengo [11] and PyNN [12] are the most used APIs to define SNN simulations. Both of them allow the description of many neurons/synapses models and can be exploited in a transparent mode on different back-ends such as neuromorphic platforms or software (SW) simulators running on general-purpose workstations.

Many research groups have developed SNN simulators to study brain functions or to develop neuromorphic applications. An updated review is provided by Carlson et al. [13]. In general, SNN simulators can be divided into two main categories: SW (digital domain) and HW (analogic, digital or mixed domains). The simulator considered in this work, namely SpiNNaker, belongs to the first category.

2.2 SpiNNaker Board

SpiNNaker is an application specific massively parallel architecture Globally Asynchronous Locally Synchronous (GALS) designed to simulate, in real-time, large scale SNN [6]. The system is built with multi-core SpiNNaker chips arranged in a two-dimensional toroidal-shaped triangular mesh (Figure 1). Each chip represents the processing nodes where neuron activities are simulated. The populations of neurons are described in SW and their spikes are represented as packets. These packets are propagated through the on-chip and inter-chip communication links via routers. Furber et al. [7] have presented a detailed description of SpiNNaker architecture. In what follows we give an overview of the key HW/SW features useful to understand the proposed methodology.

2.2.1 SpiNNaker Hardware

The SpiNNaker ***Chip*** architecture is designed in the form of a System on Chip connected to a 128MB SDRAM die, which is physically mounted on top of the SpiNNaker die and placed in the same package. In this architecture, 18 ARM968 ***Cores*** are connected through the System NoC to a custom router and to various other resources such as system ROM, System Controller, System RAM, external SDRAM and Ethernet interface (Figure 2). Each core has its own

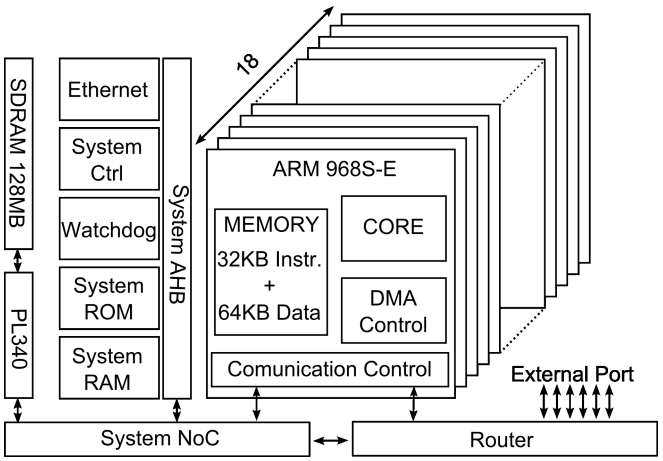

Fig. 2: **SpiNNaker chip architecture**.

DMA controller, two private tightly coupled memories for instructions and data and a bridge to the shared resources.

The ***Router*** (Figure 3) is the fundamental component of the SpiNNaker architecture, it routes the incoming packets to single or multiple outputs. Packets coming both from external links and from internal cores are presented to the router and elaborated one by one [14]. SpiNNaker chips communicate with neighbours chips via packets using the six external bidirectional links controlled by the router (Figure 3.c). Three types of packets are used during the board activity: *i) Nearest-Neighbour (NN)* for initialization and links life control at chip level; *ii) Point-to-Point (P2P)* for data communication between specific cores; *iii) Multicast (MC)* for the spike propagation during the simulation.

In order to distribute spikes across the system SpiNNaker make use of the Address Event Representation (AER) protocol [15]. When a neuron fires, a *MC* packet that contain the number that uniquely identifies the spike source neuron is generated. Then, this packet is provided on the router internal branch (Figure 3.b) and passed through multiplexer tree. At this point the packet is introduced in the second tree (Figure 3.a) that handle also the traffic coming from the six neighbour chips. The router then compares the packet identifier with the entries stored in the routing tables and in case of a match, it looks up the relative routing word. Routing word is used as selection mask that contains 1 bit for each output destination: Internal cores and external links connecting the nearest chips.

2.2.2 SpiNNaker Software

The SNN simulation is configured on the SpiNNaker system using a Python package called *sPyNNaker*. This library consists of: i) Low level software (***board SW***), written in C-ARM to be executed by the SpiNNaker board; ii) High level software (***host SW***), used to describe the SNN without any particular knowledge of the board; iii) A tool-chain used to translate the SNN described as abstract models into configuration files to be sent to the SpiNNaker cores.

The ***board SW*** is divided in three layers: The first is called *SpiNNaker Application Runtime Kernel (SARK)*, that provides low-level functions for the use of SpiNNaker chip resources. The second called *Spin1* API is running on top of SARK and implements the Event-Driven Programming

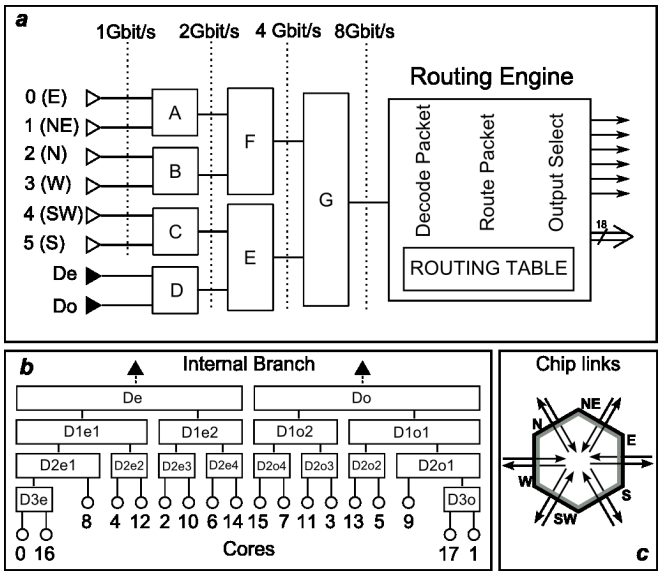

Fig. 3: **Router details**: a) the principal branch that merges the incoming packets (external and internal) to be provided to the routeing engine input; b) the multiplexer tree that connects the internal cores to the principal branch; c) the six external links of the chip.

Model used to build efficient code. The last SW level is the application program that implements one of the neuron models. These SW levels are loaded on the SpiNNaker machine during the booting phase. At the start-up, each chip runs a low level HW check. If some component is not responding it is disabled, otherwise is executed the election procedure for the monitor processor selection and the router initialization. The elected monitor processor is in charge to perform system management tasks while other cores are addressable for application processing. During the boot procedure, initiated by the host machine, SARK is loaded on all the working cores, on top of which the simulated application can be executed. The monitor processors are loaded with a special program called *SpiNNaker Control & Monitor Program (SC&MP)*, that is responsible for the supervision of chip operations and for the communications with the host computer. When *SC&MP* is loaded, each chip set its coordinate using as reference the Ethernet enabled chip that assume the $(0, 0)$ position [16].

The ***Host SW*** has been developed using of the PyNN neural system description language for the creation of an user-friendly front-end [12]. Steps in between the PyNN SNN description and its execution on the SpiNNaker board are handled by a python package called *PArtition and Configuration MANager (PACMAN)*. This package provides utilities for SNN *Partitioning*, *Placement* and *Routing* [17].

PACMAN uses the PyNN representation of SNN composed by Populations and Projections to build the *Population graph*. This graph is elaborated following three main phases. During the ***Partitioning*** phase, each neuron population is divided into portions called *part-population* in order to satisfy the core constraint of maximum number-of-neurons per core. This division is made by selecting subsets of neurons without any consideration about the neuron connectivity.

In the ***Placement*** phase, each *part-population* is assigned

to a different core by means of a simple algorithm performing the sequential positioning. Once all the cores of a chip are filled, *PACMAN* starts to fill the cores of the next chip following a radial order.

During the ***Routing*** phase the *part-populations* disposition over the board is evaluated in order to identify the best routing paths between chips. Once best paths are identified, the generation of routing tables is performed for each chip involved in the simulation.

Finally, the partitioned and placed SNN is passed to the configuration pipeline in charge to configure the SpiNNaker board with the files generated in the host.

3 Top-Down Analysis Methodology for SpiNNaker Profiling

A significant amount of research has been done to highlight the capability of simulating large SNNs on neuromorphic platforms such as SpiNNaker [18, 19]. However, the SNN behaviour is generally evaluated from a biological point of view without considering hardware faults or missing packets. Indeed, is well accepted that the SNN simulations are relatively unaffected by system variations and imperfections [20]. Profiling data of real case applications can be precious in order to improve the methods used to split the SNN populations and to decide where, on the board, is more convenient the execution of each part-population.

In a precedent work by Urgese et al. [21] a top-down SNN-based profiling methodology has been discussed. This methodology was adopted to investigate some of the SpiNNaker bottlenecks impacting on the simulation reliability and limiting the biological network size. We report here the major insight of the profiling analysis, useful to understand how the developed SNN Partitioning and Placement algorithm is able overcome the detected inefficiencies.

In order to execute an accurate profiling, we designed a customised SNN able to stimulate the main bottlenecks of SpiNNaker communication. This customized SNN is flexible enough to be used as the basic component for the design of complex use cases. One of such bottlenecks arises when a large amount of packets is transmitted in a single link at the same time in both directions with the consequent loss of packets.

The Base Configuration is built using two populations placed on two different chips (Figure 4). The first population called *Spike Source* (*SRCPop*) is used to send spikes to a connected target population following a predefined time vector. The second population, *IFPop*, is composed of *Integrate and Fire* neurons and connected one-to-one to the *SRCPop*. The behaviour of both populations is deterministic, since the *IFPop* parameters have been set to generate a new spike when a spike from the *SRCPop* is received. During the simulation, spikes generated by the *IFPop* are stored and counted in order to be compared with the number of packets generated in those cores running the *SRCPop*.

The Base Configuration is characterized by three parameters: i) The population size, responsible for the modulation of the number-of-cores used in the analysis, and the consequent number-of-packets circulating on the segments of network under test; ii) The max number-of-neurons that can be simulated over a single core; iii) The exact location of chips and cores running the two populations.

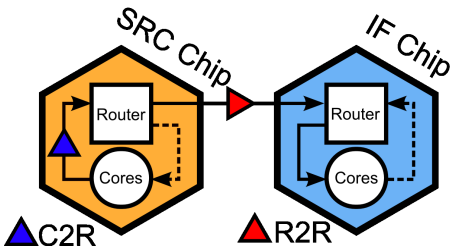

Fig. 4: **The Base Configuration (BC)**: in orange are described the *SRC Chip* while in blue the *IF Chip*. White circles represent the cores of the chips.

Using these parameters, several customised configurations can be built to force the overload of specific communication segments providing useful information about the traffic sustainability. The Base Configuration allowed us to highlight a packet loss problem occurring when all the neurons of *SRCPop* fire together and generate a huge amount of traffic over the lines of the router under investigation. In particular, we investigated two classes of traffics: The *Core to Router* traffic generated when packets come from the Cores to the Router of a chip, and the *Router to Router* traffic generated when packets are transmitted from a Router to another Router through one or more chips.

3.1 Core to Router traffic analysis

The *Core to Router (C2R)* traffic analysis is exploited to identify inefficient load configurations stressing the first internal layer of the multiplexer (Figure 3.b). This layer is in charge of introducing in the router the packets generated by the internal cores. In order to simulate a very large number of packets accessing the first internal layer of the router we adopted the Basic Configuration (in Figure 4) letting the neurons belonging to *SRCPop* to fire all together at the same time. Two *SRCPop* models have been used during the analysis: The first model exploits a software buffer to store the untransmitted packets that have to be re-introduced into the router, whereas in the second model this buffer is not used. In Figure 5 are reported four configurations (from E1 to E4) designed to show the C2R traffic response of the board with different load scenarios.

In the experiment *E1* the two populations of 4096 neurons grouped into 16 part-populations are placed on two different chips. The *SRC* neuron models without retransmission buffer are used. The neurons of *SRCPop* fire at the simulation time of 100 ms and send spikes to the connected *IF* neurons. When the *IF* neurons receive a spike they generate a new packet, and store the event. At the end of the simulation these events are counted and their occurrence compared with the number of packets conflicts collected in the *SRCPop* cores. In Figure 5.E1 are reported the configuration and the results of experiment E1. It can be observed that cores connected to the same first layer of the router internal branch (yellow and green rectangles) miss an equal amount of packet between each other; 109 packets for cores 0 and 16, while 2 packets for all the odd cores. The first hypothesis assumes that the observed behaviour is caused by bandwidth limitations on the internal router tree (Figure 3.b) even if the bandwidth reported on the datasheet (1250 packets/ms per core) is sufficient to support a

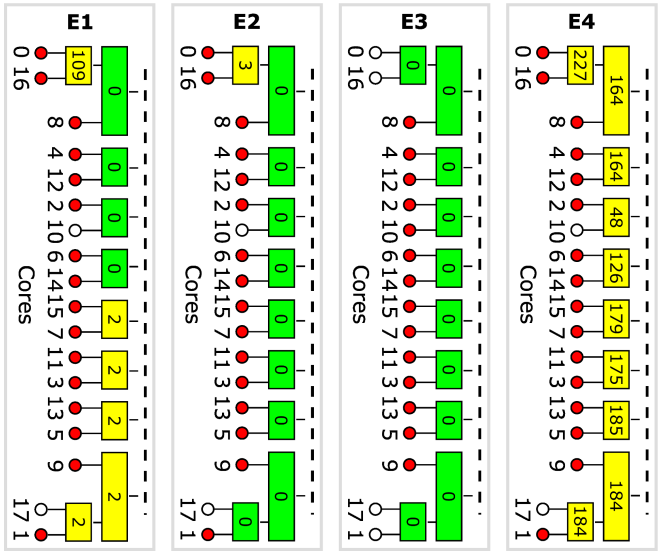

Fig. 5: **Core to Router traffic experiments.** The four experiments are represented in the router internal branch of chip executing the SRCPop. Red circles identify cores running simulation while in the connection nodes (in yellow or green) is reported the number of conflicts detected. In the experiment E 1 256 SRC neurons without retransmission buffer are placed in each of the 16 cores. Experiment E 2 is executed using only 50 neurons per core. Experiment E 3 run 256 neurons per core on 12 cores. Experiment E 4 run 256 neurons per core with retransmission buffer on 16 cores.

communication rate of 256 packets/ms per core [22], that produce an overall traffic on the router of 4096 packets/ms.

Two configurations are then executed to evaluate this bandwidth limitation hypothesis. The configuration E2 (Figure 5.E2) reduces the number of neurons per core from 256 to 50 in order to decrease the C2R traffic on the router to 800 packets/ms. The configuration E3 (Figure 5.E3) avoids concurrent packets in the first layer of router (where the majority of packet conflicts are detected), applying a delay of 1 ms to the cores 0-16 and 1-17. This configuration generates a C2R traffic on the router equal to 3584 packets/ms. Results in Figure 5.E2 shown that even in the configuration E2 some packets are dropped in the first router layer shared by cores 0 and 16. Instead, in configuration E3 all the 3584 packets were simultaneously transmitted without losses. These results highlight that conflicts are generated in the first internal router layer and are related to the *SRCPops* placement on the cores, disproving the hypothesis of router bandwidth limitation.

Finally, as last analysis, the first configuration is re-executed using the *SRC* neuron model with retransmission buffer. SRC models that make use of this buffer are able to store the conflicting packets and re-inject them as soon as possible. This re-injection system allow the correct transmission of all the 4096 generated packets. However, during this experiment we detected a higher number of conflicts with respect to the unbuffered solution (Figure 5.E4). An average of 151 conflicts per core for the buffered *SRCPops* versus 13 conflicts per core for the unbuffered version. All neuron models implement this technique and for this reason is difficult to lose internally generated packets. However, in case of congested configurations or for highly synchronous applications this solution can be time consuming. Indeed, a supplementary computational load is required from cores to support the re-transmission operations. Moreover, the adoption of these neuron models can cause premature termination of simulations due to the accumulation of delays caused by cores that are busy to retransmit packets.

3.2 Router to Router traffic analysis

The *Router to Router (R2R)* traffic analysis is designed to investigate traffic configurations that cause dropped packets in the inter-chip network. The identification of such configurations is fundamental to define reliable rules about traffic fluxes that can be used by the SNN-PP software to avoid the creation of hot spots. We designed three configuration schemes (Figure 6) to simulate traffic peaks on routers and links. In these configurations, the *SRCPops* are placed on different chips, connected to *IFpops* and configured to fire all together at the same time.

In the first configuration called *F* (Figure 6.F) the *SRCPops* are placed to transmit packets through 4 ports of the *Cross Chip*. Two out of four ports are used both as input/output: The port East(0) gets input traffic from *SRCPops A* and output traffic from the *SRCPops B*. Similarly the West(3) port sustains the traffic of the same populations with reversed input/output order. The other two ports North-Est(1) and North(2) are used in one direction only to pass packets from *SRCPops D* and *C* to the *IFPops* connected to the ports South-West(4) and South(5). During this analysis a considerable amount of packets is lost in all the routers that try to send packets through the chip under investigation. Indeed, even if 16 384 spikes are generated

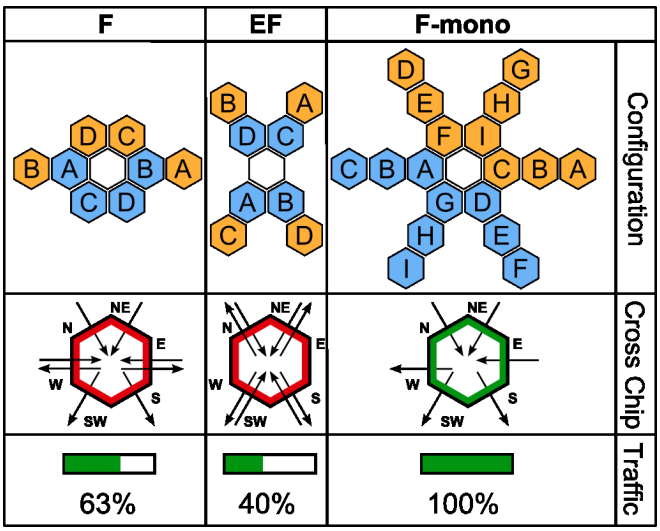

Fig. 6: **The three configurations investigated for the R2R traffic analysis**. Configurations have three main characteristics: i) the orange and blue hexagon represent the *SRCPops* and *IFpops*; ii) *SRCPops* and the connected *IFpops* are placed symmetrically in relation to the chip under investigation and marked with the same letter; iii) the hot-spot chip is represented as the white central hexagon. In the traffic section is reported the percentage of packets reaching the destinations.

by the *SRCPops* (4 *SRCPops* * 16 cores * 256 neurons-per-core) the router of *Cross Chip* processed only 10 336 packets. The 6 048 lost packets can be due to the simultaneous use of East-West communication links in both directions that generate deadlock conditions in the routers involved in the transmission path.

The configuration *EF* (Figure 6.EF) is designed to investigate the hypothesis that simultaneous bidirectional transmission from the same port can be the cause of critical traffic situations. In this configuration all the four involved ports of the *Cross Chip* are used as input/output at the same time. This configuration accounts for a higher number of lost packets with respect the F case. Indeed, only 6 508 spikes are processed by the *Cross* chip router, instead of the expected 16 384 spikes or the 10 336 packets processed by *F* configuration. Whereas, the majority of packets is dumped in the neighbour chips. In both configurations, all the routers of chips running *SRCPops* get all the packets from their cores. However, because the *Cross* chip is in a busy state, that increase when the ports are used in both directions at the same time, a deadlock chain effect is backward propagated from the busy router to the chips involved in the communication path with the relative loss of packets.

A third configuration called *F-mono* has been designed (Figure 6.F-mono) in order to validate the hypothesis that a deadlock is more likely to occur if the links are used at the same time in both directions, and to confirm that the packets loss is not due to bandwidth problems. In this configuration the traffic flows through the Cross chip in one direction only. Three *SRCPops* send packets through the three input ports of the Cross chip (Est(0), North-Est(1) and North(2)). These packets are then redirected respectively to West(3), South-West(4) and South(5) where nine *IFpops* are connected. With this configuration 36 864 spike packets are sent through the Cross chip without any loss of packets.

The use of retransmission buffer for the simulation of configurations *F* and *EF* determined a huge amount of conflicts, on average 6 conflicts per packet are generated. Furthermore, the number of links simultaneously accessed as input/output seems to impact on the number of conflicts. Indeed, 108 k conflicts were detected for the four bidirectional links configuration versus 94.5 k conflicts detected in the *F* configuration. These results demonstrate that the simultaneous communication involving opposite router links leads also to core load balancing issues.

3.3 Bio-Application Example

In order to evaluate SpiNNaker performances on real SNNs we executed the simulation of the *Cortical Microcircuit (CM)* proposed by Potjans et al. [23]. This network represents the four layers under a surface of $1mm^2$ of the human brain cortex (L23, L4, L5, and L6). Each layer consists of *inhibitory* and *excitatory* neuron populations modelled through the setting of specific parameters in the IF neuron model. Excitatory populations have positive synaptic weight while inhibitory neuron synapses are negative. The network represented in Figure 7 is described in PyNN [24]. It is composed by 77 k neurons, grouped in eight populations, and about $3 * 10^8$ synapses. Special source populations (*SRCPops*) are used to generate spikes with a Poisson probabilistic process. These

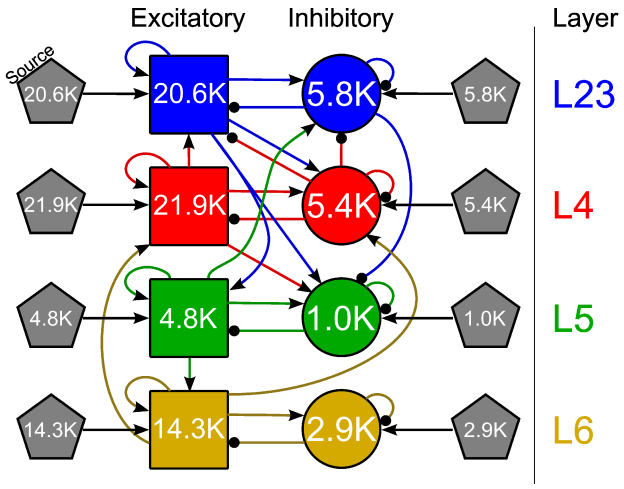

Fig. 7: **Cortical Microcircuit**: Graph representation of the SNN populations used to simulate the *CM* behaviour. The 4 layers are represented with different colours, the square represents Excitatory populations while Inhibitory are drawn as circles. The hexagons in grey stand for spike sources.

SRCPops are connected to each IF population of the *CM* to simulate the background activity of adjacent areas.

During the analysis the network has been reduced in terms of number of neurons and synapses to satisfy constraints of time and resources availability. Adopted scaling factors are in the range from 1% to 20% both for neurons and synapses. The maximum allowed neurons-per-core is an important parameter for a reliable simulation. If few neurons-per-core are set, to many cores are used and a general traffic increasing is detected in the R2R link levels. While a high number of neurons-per-core can lead to R2R traffic reduction since less cores are used, but on the other hand cores may not be able to update the neuron dynamic state in time and the C2R traffic is increased.

We run the *Cortical Microcircuit (CM)* adopting the default SNN-PP implemented in PACMAN and a customised SNN-PP method called MANUAL. The MANUAL SNN-PP forces the population placement on the board. Each chip executes the simulation of a single population, and the SRCPops are placed into the border chips close to their IFPop targets. In this way it was possible to obtain a sort of mono-directionality in the traffic fluxes between SRCPops and their IFPop targets. The *CM* simulation is executed by imposing following parameters: 5% of neurons (N05), 20% of synapses (K20), and 100 neurons-per-core. This configuration was chosen because N05 produce 3 854 *IF* neurons, the same amount of *SRC* neurons and special populations used to extend the synaptic delay called *DelayExtension*. These 11 562 neurons can be simulated over 144 cores in 10 chips with reasonable configuration and simulation time.

In Figure 8 are reported the populations arrangement on the SpiNNaker board when PACMAN and the MANUAL SNN-PP procedures are adopted. The chips are coloured with the same colours used to represent the populations in Figure 7. Moreover, the overall data traffic collected by the router counters during one second of simulation is reported, together with the missed spikes. In the first *CM* simulation, when PACMAN is adopted, a total of 723 packets have been

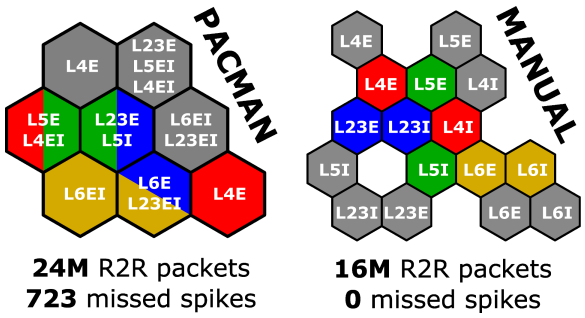

Fig. 8: **Placement of CM network**: each hexagon represents a SpiNNaker chip. The colour represents the *CM* population executed in each chip, in grey chips that contains the *SRCPops*.

dropped. Whereas, the R2R packets characterizing the on board traffic are about 24M. Even if the number of dropped packets can be considered very low with respect to the overall number of circulating packets, it is a good practice to ensure the correct transmission in order to prevent unreliable simulation results. In the second *CM* simulation, the populations are mapped on the board applying the MANUAL SNN-PP algorithm. On the right side of figure 8 it is shown this positioning where a full chip is used to run a single population. In order to optimize the packets flow among the chips, *SRCPops* are executed in the perimeter chips while other populations are placed in the middle. The use of MANUAL mapping procedure, that prevented sub-optimal configurations, is useful to reduce the number of R2R packets and to eliminate the dropping events. Indeed, this customized procedure produces a reduction of 33% of the number of R2R packets, from 24 M to 16 M.

4 SNN Partitioning and Placement

We studied the SNN Partitioning and Placement (SNN-PP) algorithms currently used in PACMAN (see Section 2.2.2) and we noted that none of the highlighted problems were considered during these procedures. Two considerations should be made on the SNN-PP procedure implemented by PACMAN. The *Basic Partitioning* algorithm is very naive, it splits the populations considering as the only constraint the neurons-per-core that a single core can simulate. Populations with a number of neurons exceeding this threshold are sequentially split to create part-populations of the correct size. In the worst cases the last part-population can be created even for a single neuron, wasting the resources of a core while keeping all the others saturated.

The second consideration concerns the PACMAN SNN placer. The *Radial Placer* algorithm maps the part-population without considering the effective network connectivity with the risk to place far away two highly connected groups of neurons. The high flow of packets inside the network, caused by naive part-populations placement, can be the cause of the hot-spot creation with consequent packet traffic unreliability. Using these two considerations as starting point, we present in the following our SNN-PP method that improves the performance and the reliability of these two important steps.

Placement techniques are extensively studied in the VLSI field where the component graph is positioned in the chip area using the design constraints. These techniques are typically based on a *Simulated Annealing*, *Analytical Paradigms*, or *Partition Based* approaches. The aim of *Partition Based* approaches is to minimize the mutual connections and keep close the highly connected components. This class of algorithms is often preceded by several clustering phases aimed at reducing the dimensionality of the networks and for this motivation is commonly referred as *Multilevel Partition Based Approach (MPBA)* [25].

The SpiNNaker cores can simulate at most a number-of-neurons, belonging to the same populations, proportional to the complexity of used neuron model (upper limits is 256). For this motivation, when a SNN graph is placed on the SpiNNaker cores, it is necessary a partition procedure able to split the populations with a number-of-neurons greater than the core upper limit.

In literature a *Simulated Annealing* approach has been theorised by Brown et al. [26] to find a solution to the SNN partitioning and placement (SNN-PP) problem. However, at the state-of-the-art none implementation can be found. We studied the three main placement techniques adopted in the VLSI field and identify the MPBA class as the most compliant to solve the SNN-PP problem. Indeed, the SNN partitioning procedure can be natively supported by partition based techniques. The *MPBA* can be used such a starting point for the development of an ad-hoc methodology to solve the SNN-PP problem on the SpiNNaker board. The graph representing SNN can be partitioned and placed such as the graph of VLSI components. The major difference is represented by the SNN huge fan-in to be handled (up to 10k synapses per neuron).

In order to design an optimised SNN-PP method capable to assign the neuron populations to the SpiNNaker cores, we exploited the evidences identified during the profiling analysis. We identified two major aspects to be considered during the SNN-PP: i) The physical position, in the chip architecture, of cores running the populations has an effect on the efficiency of the packet transmission. The load balancing of the internal multiplexer tree can be a good practice to reduce the overload due to packet conflicts. ii) Bidirectional traffic involving opposite ports of a chip can lead to deadlock conditions that cause dumping events with relative loss of packets.

In biologically meaningful cases links cannot be used in a mono-directional mode because of SNNs complexity. Therefore, it has been studied an SNN-PP algorithm able to minimize the packet flows in the inter-chip network (R2R) without trying to force mono-directional traffic. This algorithm, called SNN Spectral Analysis based Partitioning and Placement (SNN-SA-PP), is applied to the graph representations of the analysed SNN.

SNNs are represented through three graph layers, each of them useful for the execution of specific operations: i) *Population Graph* is the representation where each vertex is a PyNN Population and each edge is a Projection with a connector for the synapses generation; ii) *Neuron Graph* is a SNN representation where each vertex is a neuron and each edge is a synapse; iii) *Part-population Graph*, generated from the Neuron Graph clustering, is the representation where each edge is a set of synapses and each vertex is a part-population with a number of neurons that can fit in a core.

The main processing flow (shown in Figure 9) takes as input the *Population Graph*, removes the *SRCPops* that will be partitioned and placed at the end of the procedure, and expands this graph in order to get the *Neuron Graph*.

During the *Partitioning* phase, this detailed graph is analysed using spectral clustering techniques. The generated clusters, of predefined neuron size, are then used to create the *Part-population Graph*. The spectral analysis applied to the *Neuron Graph* allows to label each neuron with a n-dimensional coordinates, in this way neurons can be managed like points where the distance between two of them represent their connectivity. By applying the clustering algorithm is then possible to isolate sets of neuron highly connected and to map them together. Thus, the partitioning problem can be solved iteratively through the *Sub-Clustering* phases, where neurons from the same population and cluster are grouped in sub-clusters matching the core constraints (neurons-per-core).

Vertexes of *Part-population Graph* are generated using the centroids of sub-clusters. Thus, each vertex of this graph represents a sub-cluster. Moreover, in order to prevent the generation of small part-populations, that lead to unoptimised use of cores, a *Fusion* phase is executed where sub-clusters are analysed and in some cases manipulated.

For the *Placement* phase the part-populations graph is elaborated using the *Sammon Mapping* multidimensional scaling algorithm. This scaling procedure is applied in order to adapt the graph multidimensionality in a bi-dimensional space. This representation can be used for a direct placement process where a rectangular grid is build upper the points and each area is associated with a chip. During the *Sammon Mapping* part-populations that fall in an area are mapped in the free cores of the chip. If the number of part-populations in a single chip/square exceed the number of available cores a *Legalization* procedure is applied where a simple greedy algorithm move the extra point in the free nearest areas. In the example reported in *Figure 9* the constraint is one part-population for each chip, whereas in real cases up to 16 part-populations can be placed. Placement procedure ends with the assignation of the *SRCPops*, extracted in the first phase of SNN-SA-PP, to each chip running the associated part-populations. Space can be reserved in each chip for this type of population accordingly to their particular connectivity. At last step, the configuration files are generated and sent to the SpiNNaker board.

We implemented the SNN-SA-PP method in a Python module called GrapH Optimizer SpiNNaker Tool (GHOST). This SW layer exposes a PyNN front-end and provides a pre-partitioned and pre-placed population graph to the sPyNNaker library that generate the configuration files to be sent to SpiNNaker for the simulation.

4.1 Partitioning

In order to design an efficient Partitioning algorithm we adopted a *Multilevel Partition Based Approach (MPBA)*. Two clustering algorithms were identified such as easily applicable to the graph partition problem: The *Highly Connected Sub-graphs (HCS)* and the *Spectral Clustering*. The *HCS* algorithm implements recursively bi-partition steps over the input graph until the sub-graphs reach a predefined

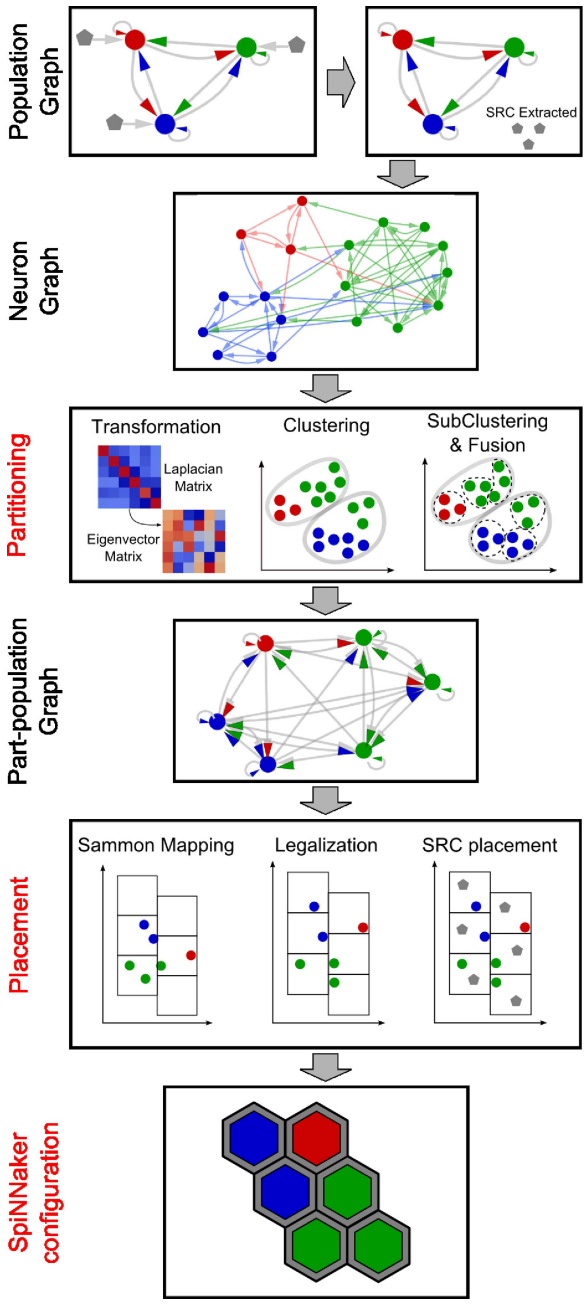

Fig. 9: **Flowchart of SNN Spectral Analysis based Partitioning and Placement algorithm**. From SNN Population Graph to the chip placement of neurons.

degree of connectivity [27]. The definition of the degree of connectivity for the termination condition and the choice of the method to be used to execute the bi-partition steps are the major problems of this method. These problems are avoided if the Spectral Clustering techniques are adopted. The Spectral Clustering method makes use of the eigenvectors of the similarity matrix associated to the input graph to obtain an optimised spatial disposition of the data [28]. Applying this method on the *Neuron Graph* we obtained an optimised spatial disposition of the neurons useful for both Partitioning and Placement phases.

The *Population Graph* is transformed into a more detailed *Neuron Graph* representation extracting the neuron grouped in the populations and generating all the synaptic connec-

tions. The *Neuron Graph* is a directed graph $G(N, S)$ where N and S are respectively the neurons and the synapses sets. Each synapse $s \in S$, is defined with four parameters: The source neuron i, the target neuron j, the synaptic weight w_{ij} and the synaptic delay d_{ij}.

An undirected graph can be represented with the *Adjacency* matrix $A(i,j) = 1\, if\, w_{ij} > 0$. This matrix is then elaborated in order to obtain the *Affinity* matrix $L = D–A$ where D is the *Degree* matrix (a diagonal matrix containing the degree of each vertex). In our case, the direct graph elaboration required a more complex procedure. This procedure starts with the computation of the *Normalized Laplacian* matrix of G defined in Equation 1, where I represents the *Identity* matrix, P is the *Transition* matrix of the graph G induced by a random walk method and Φ is the matrix with the Perron vector of P in the diagonal and zeros elsewhere [29].

$$L = I - \frac{(\Phi^{\frac{1}{2}} P \Phi^{-\frac{1}{2}} + \Phi^{-\frac{1}{2}} P^T \Phi^{\frac{1}{2}})}{2} \tag{1}$$

Using L like *Affinity* matrix we compute eigenvalues and eigenvectors. Each neuron can be represented in u dimensions of the spectral space using the first u eigenvectors sorted from the largest to the smallest eigenvalues. The number of dimensions u, is fixed such as $u = k$ where the k parameter represents the number of clusters to be imposed to the first step of the clustering algorithm. The k parameter can be calculated using the Equation 2, where N is the number of neurons to be simulated, c represent the available cores on each chip and n is the maximum number of neurons that can be executed on each core.

$$k = \lceil \frac{N}{c * n} \rceil \tag{2}$$

The k-Means clustering procedure is at this point used to transform the *Neuron graph* into the *Part-population Graph*. The objective function in Equation 3 aims to minimize the distance between the x points of the cluster S with respect to the cluster centre ϕ.

$$\min \sum_{i=1}^{k} \sum_{x \in S_i} \| x - \phi_i \|^2 \tag{3}$$

Clustering is executed in three steps: i) k cluster centroids are chosen randomly; ii) for each iteration, points are assigned to the closest centroid and the new cluster centroid is updated; iii) computation ends when the difference between the objective function value in two successive iterations is lower than a fixed threshold. Concluded the clustering procedure, each cluster is analysed and divided in part-populations. Neurons belonging to the same population are grouped to create sub-clusters of size compatible with core constraints. The centroids of clusters are then used as vertices of the SNN *Part-population Graph* representation. Three methods can be used to make this intra-cluster division: Random neuron division, position based neuron division, or using a balanced sub-clustering method. For the Partitioning algorithm here reported has been used the k-Means algorithm together with a heuristic voted to balance the number of neurons for each sub-cluster.

During the cluster division phase the *Fusion* procedure is executed in order to prevent the generation of small part-populations that lead to unoptimised use of resources and to increase the packet traffic. If neurons belonging to the same population are split in more than one cluster and one of them has less than 20% of neurons that a core can simulate, these neurons are moved into the nearest cluster. In Figure 10 a simple example is shown, where a neuron belonging to the green population is clustered with neurons of the blue population. The fusion procedure recognises the green neuron and reassign it to one of the green sub-clusters, avoiding the creation of a supplementary vertex into the *Part-population Graph*.

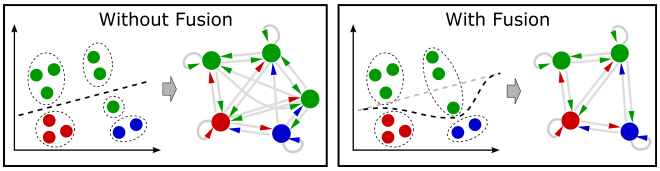

Fig. 10: **Fusion step.** This procedure allows merging uncompleted clusters in order to optimise the resources use.

4.2 Placement

The part-populations placement on the SpiNNaker cores is made by considering the coordinates in the eigenvectors space obtained during the Partitioning. Since part-populations have coordinates in k dimensions, a dimension reduction process is applied to place them on the planar hexagonal mesh formed by SpiNNaker chips. This process can be done by using the first two eigenvalues only or by applying an algorithm to calculate a more accurate multidimensional scaling. The application of a multidimensional scaling algorithm on the coordinates of points allows to reduce the dimensionality of points taking into account the distances in the k dimensional space. In the SNN-SA-PP this task is done using a *Sammon Mapping* algorithm [30]. This algorithm uses a non-linear approach to minimize the relative distance d of the high dimensional points with respect to the relative distances of the points with low dimensionality (Equation 4).

$$E = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}} \tag{4}$$

Using the Equation 4 we map the points in the *Sammon space*, a bi-dimensional space populated by the part-populations to be placed on the SpiNNaker architecture. In order to exploit the spatial information of part-populations during their placement over the chips, we divided the *Sammon space* applying a rectangular grid with shifted columns (placement step in Figure 9). This grid has z rows and z columns with $z = \lceil \sqrt{p/c} \rceil$ where p is the number of part-populations and c is the max number of part-populations for each chip. Even columns are vertically moved for mimic a hexagonal shape, in this way each rectangle is in touch with six neighbours and represents a SpiNNaker chip.

All points that fall in a rectangle should be mapped on a core of the relative chip. Since more points than available cores can fall into the same square a *Legalization* step is executed to move the extra part-populations to the free neighbours chips. This procedure is implemented as a

greedy algorithm that move extra points to the free nearest areas (as shown in Figure 9 at Placement step).

At last step, the *SRCPops* isolated in the early phase are directly placed on the reserved cores. The spike source neuron models (SRC) are used in SNNs to start or maintain special regimes of activity. For example, in the Cortical Microcircuit analysed in Section 3.3, the SRC neurons are used to simulate the background activity of adjacent cortical areas. SRC neuron is usually connected to a single target neuron and configured to simulate a high level of activity generating an intense traffic of packets. For these motivations placing SRCPops on the same chips that host the target neurons resulted as a good practice to reduce the R2R traffic.

5 Results

The validation process adopted to demonstrate the improvement obtained using the SNN-SA-PP algorithm is here discussed, together with the achieved results. Three Partitioning and Placement variants are adopted for this purpose. i) The *No-Fusion* make use of the SNN-SA-PP algorithm described in Section 4 with the exception of the fusion step that is not implemented. ii) The *Fusion* uses the full procedure SNN-SA-PP shown in Figure 9 to transform the Population graph into an optimized SpiNNaker configuration. iii) The *Random* is used as a reference to validate the improvement obtained by the other two SNN-SA-PP variants. It makes a random division of *Neuron graph* considering only the number-of-neurons per part-population that must be kept homogeneous. Whereas, it applies a radial placement of the IF part-populations keeping in the same chip the associated SRCPops and DelayExtension.

The Cortical Microcircuit (CM) has been used to demonstrate the effectiveness of SNN-SA-PP method implemented in GHOST and to compare the achieved results with those obtained by the use of the SNN-PP implemented in PACMAN. In order to be compliant with the experiment proposed in Section 3.3 the scaling factor N05-K20 has been adopted. Moreover, in order to observe the system response when synapses number decrease from 3 M (20%) to 750 k (5%) we used a second scaling factor equal to N05-K05.

Six rounds of simulations were executed to extract the ratio R2R/C2R and the overall R2R packets circulating in the network. In each round, 20 simulations were performed using one of the three SNN-PP variant (Fusion, No-Fusion or Random) to set-up the SpiNNaker board with one of the two scaled CM. R2R/C2R ratio represents the traffic circulating in the inter-chip network versus the traffic generated into the chips. This ratio is used to compare the performances of the three investigated SNN-PP variants. Lower values of this rate correspond to the capacity of the network to keep local the communications, reducing the number of packets circulating on the inter-chip links.

As can be seen in Table 1, the N05-K05 SNN placed with the SNN-PP *Random* variant generates an average of 292 k R2R packets. Instead, the network scaled at N05-K20 generates an average of 323 k R2R packets. In Figure 11 it is shown as CM configurations generated using the SNN-PP *Random* variant produces small fluctuations on the R2R/C2R ratio, with all the values concentrated near 9.10E-2 for the N05-K05 case and 2.55E-2 for the N05-K20.

Partitioning & Placement		Packets R2R [k]		
CM config.	Version	Worst	Average	Best
N05-K05	Random	302	292	283
	No-Fusion	378(+25%)	318(+9%)	231(-18%)
	Fusion	265(-12%)	237(-19%)	228(-19%)
N05-K20	Random	331	323	309
	No-Fusion	411(+24%)	264(-18%)	245(-21%)
	Fusion	289(-13%)	252(-22%)	238(-23%)

TABLE 1: **Cortical microcircuit R2R traffic** detected for two configurations N05-K05 and N05-K20 when three Partitioning and Placement variants of proposed technique are applied: Random, No-Fusion and Fusion. The R2R packets percentage is computed comparing Spectral and Fusion variant with the Random configuration.

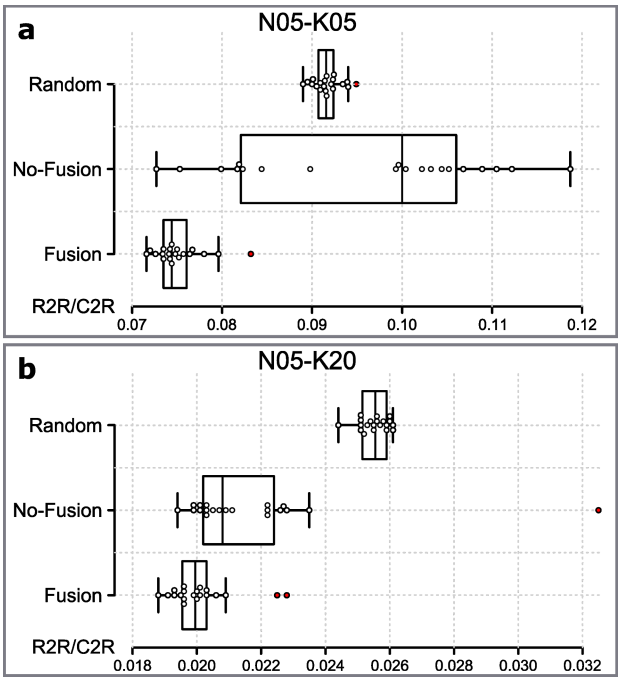

Fig. 11: **Distribution of R2R/C2R ratio** computed on a two configurations N05-K05 and N05-K20. 20 simulations have been performed for the three SNN-PP techniques.

The executions of N05-K05 configured with SNN-PP *No-Fusion* variant produced a larger range of R2R/C2R ratio (Figure 11.a). Indeed, if compared with the *Random* variant, only part of the experiments come out with more balanced configurations. This was due to the generation of small part-populations (less than 10 neurons) that lead to the use of supplementary cores, thus increasing the R2R traffic.

To prevent this unwanted behaviour, in the SNN-PP has been added the Fusion step that is able to solve this problem. Experiments performed using the SNN-PP *Fusion* variant demonstrate that the R2R/C2R rate is always lower than the rate produced by the *Random* variant. Considering the N05-K20 we note that even the R2R/C2R rate of SNN-PP *No-Fusion* variant is always better than the rate of *Random*. This is principally due to the higher number of connections that

produce more distant points for the clustering algorithm that can better balance the clusters and make negligible the generation of small clusters.

The average number of R2R packets produced by N05-K20 placed on SpiNNaker using the SNN-PP *No-Fusion* is reduced of 60 k packets with respect to those placed with the *Random* variant (Table 1). This is not true for N05-K05 where the influence of small part-population affects the average. Indeed, an increase of 26 k R2R packets is detected when the *No-fusion* variant is used. CM configurations generated with *Fusion* variant give always better balanced traffic and less R2R packets than the *Random* and *No-fusion* variants. The *Fusion* worst case with 265 k and 289 k packets is better than *Random* best case of 20 k R2R packets (-6% of R2R packet). In average, the *Fusion* variant decreases the R2R packets of 55 k in N05-K05 and 70 k in CM N05-K20 (20% of R2R packet less than *Random* variant).

First and third quartiles of Fusion box plots in Figure 11 confirm that the spectral analysis is a suitable technique that applied to the SNN-PP problem within SpiNNaker can produce good results in reducing the inter-chip traffic. The effect is increased if associated with a fusion system capable to avoid the generation of little Part-populations.

At last analysis step, we propose a comparison between CM Partitioned and Placed using PACMAN and the configuration produced by our SNN-SA-PP algorithm implemented in GHOST. As it is possible to note in Figure 12 the 24 M of R2R packets generated by CM placed with PACMAN are reduced to 250 k if CM is configured with GHOST allowing a 96X reduction of R2R traffic.

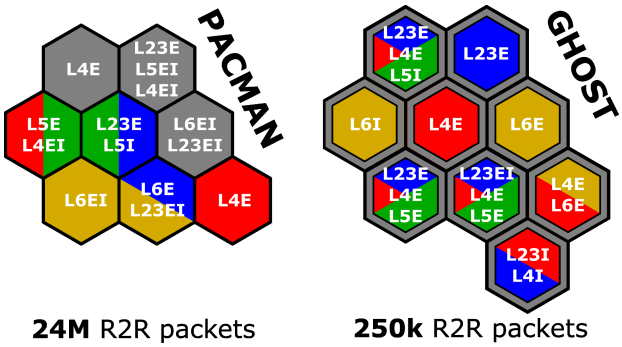

Fig. 12: **The placing of CM network**: Each hexagon represents a SpiNNaker chip. The colours represent the CM populations inside the chips. In grey the chips that run the SRCPops.

6 Conclusions

In this paper we described a methodology to efficiently map spiking neuron populations in neuromorphic platforms for reliable and scalable SNN simulations. We applied our methodology to SpiNNaker, a densely interconnected neuromorphic multi-chip many-core platform for real-time SNN simulations. We first described a top-down profiling analysis performed to characterise reliability issues in the SpiNNaker platform. This profiling methodology was designed in order to highlight the impact of neuron population mapping on the optimal platforms resource exploitation. We discussed the profiling methodology based on the use of custom SNN configurations that revealed both Core-to-Router and Router-to-Router traffic issues. The Core-to-Router traffic analysis was useful to detect packet conflicts in the internal router tree related to traffic congestion caused by the high contemporaneity and intensity of generated spikes. While exploiting the Router-to-Router analysis, we found out that the simultaneous occupation of links in both directions is a potential source of unreliability.

A simulation of a Cortical Microcircuit SNN was evaluated to characterise unreliable SpiNNaker behaviours such as packet losses in communication links and simulation failure when the PACMAN SNN-PP algorithm is used. In the second part of the paper, we focused on the cluster-based SNN-PP algorithm we developed where SNNs represented as *Population Graphs* are exploded in *Neuron graphs* and clustered using Spectral Clustering techniques. Isolated clusters were then arranged in a *Part-population Graph* and placed on the available cores and chips using the information collected during the clustering step. This method was implemented in a Python module compliant with the PACMAN tool chain, called GHOST.

A Cortical Microcircuit was simulated again with two scale factors in order to demonstrate the effectiveness of the developed SNN-PP cluster approach with respect to random neuron placement. Moreover, from these simulations was evident that the Fusion procedure, which is capable to reduce the number of used cores, results in lower R2R traffic.

Finally, comparisons were made between configurations produced by PACMAN SNN-PP algorithm and the SNN-SA-PP implemented in GHOST. These comparisons showed a R2R traffic reduction of 96X when GHOST is adopted. In future works we plan to upgrade the SNN-PP algorithm phases by expanding the clustering and legalization steps. In addition we plan to improve the portability of the algorithm in order to give support to alternative SNN description languages used in different neuromorphic platforms.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme ([FP7/2007-2013]) under grant agreement no 604102 (HBP). We also thank the APT Group of Manchester University for providing us the SpiNNaker board, and in particular Prof. S. Furber and Mr. A. Rowley for their discussions and constructive suggestions.

References

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] D. F. Goodman and R. Brette, "Brian simulator," *Scholarpedia*, vol. 8, no. 1, p. 10883, 2013.

[3] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[4] H. Markram, "The blue brain project," *Nature Reviews Neuroscience*, vol. 7, no. 2, pp. 153–160, 2006.

[5] K. Minkovich, C. M. Thibeault, M. J. O'Brien, A. Nogin, Y. Cho, and N. Srinivasa, "Hrlsim: a high performance spiking neural network simulator for gpgpu clusters," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 25, no. 2, pp. 316–331, 2014.

[6] S. B. Furber, F. Galluppi, S. Temple, and L. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[7] S. B. Furber, D. R. Lester, L. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *Computers, IEEE Transactions on*, vol. 62, no. 12, pp. 2454–2467, 2013.

[8] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM*, vol. 57, no. 6, pp. 13–15, 2014.

[9] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain research bulletin*, vol. 50, no. 5, pp. 303–304, 1999.

[10] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[11] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, 2013.

[12] A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in neuroinformatics*, vol. 2, 2008.

[13] K. D. Carlson, M. Beyeler, N. Dutt, and J. L. Krichmar, "Gpgpu accelerated simulation and parameter tuning for neuromorphic applications," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 570–577.

[14] S. Furber, S. Temple, and A. Brown, "On-chip and inter-chip networks for modeling large-scale neural systems," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*. IEEE, 2006, pp. 4–pp.

[15] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 47, no. 5, pp. 416–434, 2000.

[16] T. Sharp, C. Patterson, and S. Furber, "Distributed configuration of massively-parallel simulation on spinnaker neuromorphic hardware," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 2011, pp. 1099–1105.

[17] F. Galluppi, S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber, "A hierachical configuration system for a massively parallel neural hardware platform," in *Proceedings of the 9th conference on Computing Frontiers*. ACM, 2012, pp. 183–192.

[18] A. D. Rast, F. Galluppi, X. Jin, and S. Furber, "The leaky integrate-and-fire neuron: A platform for synaptic model exploration on the spinnaker chip," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[19] T. Sharp, R. Petersen, and S. Furber, "Real-time million-synapse simulation of rat barrel cortex," *Frontiers in neuroscience*, vol. 8, 2014.

[20] J. Navaridas, S. Furber, J. Garside, X. Jin, M. Khan, D. Lester, M. Luján, J. Miguel-Alonso, E. Painkras, C. Patterson, L. A. Plana, A. Rast, D. Richards, Y. Shib, S. Temple, J. Wue, and S. Yangf, "Spinnaker: fault tolerance in a power-and area-constrained large-scale neuromimetic architecture," *Parallel Computing*, vol. 39, no. 11, pp. 693–708, 2013.

[21] G. Urgese, F. Barchi, and E. Macii, "Top-down profiling of application specific many-core neuromorphic platforms," in *Embedded Multicore/Manycore SoCs (MCSoC), 2015 IEEE 9th International Symposium on*. IEEE, 2015.

[22] S. Furber, "Spinnaker - a chip multiprocessor for neural network simulation. datasheet. v2.02," 2011. [Online]. Available: https://solem.cs.man.ac.uk/documentation/datasheet/SpiNN2DataShtV202.pdf

[23] T. C. Potjans and M. Diesmann, "The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model," *Cerebral cortex*, vol. 24, no. 3, pp. 785–806, 2014.

[24] B. S. Bhattacharya, C. Patterson, F. Galluppi, S. J. Durrant, and S. Furber, "Engineering a thalamo-cortico-thalamic circuit on spinnaker: a preliminary study toward modeling sleep and wakefulness," *Frontiers in neural circuits*, vol. 8, 2014.

[25] C. Chu, "Placement," *Electronic Design Automation: Synthesis, Verification, and Testing*, pp. 635–684, 2007.

[26] A. Brown, D. Lester, L. Plana, S. Furber, and P. Wilson, "Spinnaker: The design automation problem," in *Advances in Neuro-Information Processing*. Springer, 2009, pp. 1049–1056.

[27] E. Hartuv and R. Shamir, "A clustering algorithm based on graph connectivity," *Information processing letters*, vol. 76, no. 4, pp. 175–181, 2000.

[28] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[29] F. Chung, "Laplacians and the cheeger inequality for directed graphs," *Annals of Combinatorics*, vol. 9, no. 1, pp. 1–19, 2005.

[30] J. W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Transactions on computers*, no. 5, pp. 401–409, 1969.

Gianvito Urgese is a Ph.D. student at the Dept. of Control and Computer Engineering of Politecnico di Torino. He received his M.Sc. degree (summa cum laude) in Electrical Engineering at Politecnico di Torino. He designed, during his M.Sc. thesis, an optimised HW accelerator for sequence alignment, implemented in VHDL on a systolic array architecture. He was, in 2011, research trainee at Teseo S.p.A. involved in the design of a system for structural health monitoring in composite materials. In 2008 he collaborates with the INRIM institute for a project concerning the redefinition of Boltzmann constant. His research interests focus on: (i) Research and design of optimized task-specific bioinformatics algorithms; (ii) Development of tools for the study of non-coding biological sequences (miRNA, siRNA and lncRNA); (iii) Design of heterogeneous SW/HW architectures to accelerate bioinformatics algorithms, including parallel implementation on FPGA and GPU; (iv) Design of partitioning and placement algorithms to map Spiking Neural Networks in the SpiNNaker neuromorphic platform.

Francesco Barchi is a research assistant at the Dept. of Control and Computer Engineering of Politecnico di Torino. He received his M.Sc. degree in Computer Engineering at Politecnico di Torino. He has experience in multidisciplinary tasks with a collaboration with GAMUT s.r.l for the development of a MASW analysis software, and he designed, during his M.Sc. thesis, an optimised technique for partitioning and placement of SNN in the SpiNNaker neuromorphic platform. His research interests focus on optimisation problems and software develop for bioinformatics algorithms: MD, SNN and sequence alignment on heterogeneous platforms, from GPU to many-chip multi-core architecture.

Enrico Macii is a Full Professor of Computer Engineering at Politecnico di Torino. From 1991 to 1997 he was also an Adjunct Faculty at the University of Colorado at Boulder. He holds a PhD degree in Computer Engineering from Politecnico di Torino (1995). Since 2007, he is the Vice Rector for Research and Technology Transfer at Politecnico di Torino, and since 2012 also the Rector's Delegate for International Affairs. He was the National FP7 ICT Delegate from 2011 until 2013, and one of the Italian Members of the Public Authorities Board of the ENIAC and ARTEMIS Joint Undertakings from 2009 until 2013. His research interests are in the design of electronic digital circuits and systems, with particular emphasis on low-power consumption aspects. In the field above he has authored around 450 scientific publications.

Andrea Acquaviva is Associate Professor at Politecnico di Torino, Italy. He received the Ph.D. degree in electrical engineering from the University of Bologna, Italy, in 2003. In 2003, he became an Assistant Professor in the Computer Science Department, University of Urbino, Italy. From 2005 to 2007, he was a Visiting Researcher in the Ecole Polytechnique Federale de Lausanne, Switzerland. In 2006, he joined the Department of Computer Science, University of Verona, Italy. He has been with the Department of Computer Engineering and Automation, Politecnico di Torino. His research interests focus mainly on parallel computing for distributed embedded systems such as multi-core and sensor networks and simulation and analysis of biological systems using parallel architectures. In the fields above, he has authored over 140 scientific publications.