Directed Graph Placement for SNN simulation into a multi-core GALS architecture

# Directed Graph Placement for SNN Simulation into a multi-core GALS Architecture

Francesco Barchi, Gianvito Urgese, Andrea Acquaviva, and Enrico Macii
Department of Control and Computer Engineering
Politecnico di Torino
Torino, Italy 10129
francesco.barchi@polito.it

***Abstract*—In this paper, we present a methodology for efficiently mapping neural networks over a neuromorphic computing architecture. The target architecture is a globally asynchronous locally synchronous (GALS) multi-core designed for simulating spiking neural networks (SNN) in real-time, that is spike timings should be the same as in the human brain. The SNN is implemented as a set of concurrent tasks modelling the behaviour of biological neurons, which are executed on the processing cores and communicate through spikes travelling on a network-on-chip. The problem of neuron-to-core mapping is relevant as a non-efficient allocation may impact real-time and reliability of the neural network execution. We designed a task placement pipeline capable of analysing the network of neurons and producing a placement configuration that enables a reduction of communication between computational nodes. The neuron-to-core mapping problem has been formalised as a problem of minimisation of synaptic elongation. Intuitively, this metric represents the cumulative distance that spikes generated by neurons running on a specific core have to travel to reach their destination core. The proposed placement methodology allows using different techniques to solve the problem. In this work Spectral Analysis, Multilevel Static Mapping, and Simulated Annealing were compared evaluating the overall post-placement synaptic elongation. Results point out that mapping solutions taking into account the directionality of the SNN provide a better placement and quantify this impact. Between all techniques considered only the Simulated Annealing was able to overcome an improvement of 25% compared to a random placement.**

## I. Introduction

Finding the best way to map processes to physical cores (PCs) in multi and many-core systems is a relevant optimisation problem, with significant impact on application reliability, performance, and energy consumption. We explored this problem in the case of a globally asynchronous locally synchronous (GALS) multicore architecture designed for neuromorphic applications. Here, tasks to be executed are physical neuron models running in parallel on the platform and communicating through messages. These messages represent signals, called spikes, which biological neurons exchange through their physical (neural) connections inside the brain.

The overall purpose of this application is to execute a Spiking Neural Network (SNN) in real-time. In this case, real-time means that the timings of the spikes generated by the neurons should be compliant with the one of the real human brain. Thus opening the way for the use of neuromorphic platforms to interface external physical systems and elaborates their signals (e.g. images, sounds) in the same way as the brain does. Being the neurons executed as concurrent tasks by the general purpose cores, there is a problem of efficient mapping of neurons to cores to optimise the communication between them. Indeed, these tasks are characterised by intensive communication activity.

Generalising, the problem we faced concerns the mapping of a large number of light parallel tasks with intensive communication to a many-core architecture. A non-efficient communication, in the specific case of SNN execution, may impact real-time capabilities as well as the reliability of the application. Indeed, spikes can be lost due to congestion problems. In general, a possible approach to face the mapping problem is to model the tasks and their communication as a graph to be mapped over the underlying hardware architecture, represented by another graph.

In this paper, we present a methodology for mapping a task graph representing the SNN computation on a multi-chip many-core architecture with communication awareness. To achieve this target, we designed a task mapping framework capable of analysing the network of neurons to find a configuration with the target of reducing the communication between computational nodes. The neuron-to-core mapping problem has been formalised as a problem of minimisation of synaptic elongation. Intuitively, this metric represents the cumulative distance that spikes generated by neurons running on a specific core have to travel to reach their destination core.

The framework starts by extracting a graph of independent processes from a neural network description. In the case of SNN, the direction of a communication path is also to be represented using a directed graph. On the platform side, the interconnect structure is described as a graph where nodes represent on-chip cores while edges represent physical communication links between them. In this way, we formalised a neuron-to-core mapping as a graph-matching problem solvable through the exploitation of various algorithms available in the literature. The specific formulation we devised for SNN mapping takes into account the typical organisation of these type of neural networks into neuron populations, sharing similar characteristics as well as the neuron model.

The results obtained by comparing four mapping algorithms

978-1-5386-4756-1/18/$31.00

points out and quantify the relevance of the communication direction information to achieve a better mapping if compared with non-directional algorithms.

The paper is organised as follows. In Section II we give a brief overview of the application and the target board used in this work. In Section III we detailed the problem formulation under the graph theory view-point, while we describe our process placement method in Section IV. In Section V we report on the results obtained during the validation process performed on the cortical microcircuit SNN simulation. Finally in Section VI we reach some conclusions and list some future declinations of this piece of work.

## II. Background

In this section, we will introduce the application and the MCSoC board selected as a target for demonstrating the advantages of adopting our process-placement communication aware framework.

Spiking Neural Network (SNN) is a particular neural model used by neuroscientist for simulating biologically plausible brain activity. During SNN simulations neurons and their synapses are modelled as differential equations capable of emulating the behaviours observed in biological networks [5]. Two of the most adopted neuron models are the *Leaky Integrate and Fire* (LIF) [3] and *Izhikevich* (IZK) [4], because they are able to ensure a plausible picture of the biological behaviours with reduced computational costs. Van Albada et al. [7] designed an SNN application implementing the cell-type specific *cortical microcircuit* (CM) model created by Potjans et al. [8]. Then they simulated this SNN on a neuromorphic multi-chip many-core platform called SpiNNaker [9] using the standard application partitioning and placement system for setting up the simulation on the board.

For validating our placement methodology framework, we took as target a GALS Neuromorphic many-core architecture and used its native application such as an example case. We used the SpiNNaker architecture [9], which is a general-purpose real-time many-core platform mainly used for simulating neural networks following an event-driven computational approach but the methodology could also be applicable for the new Loihi platform [**?**] and the future SpiNNaker 2 architecture.

The SpiNNaker chip has 18 ARM 968 cores running at 200MHz, a full-custom router for intra/inter-chip communications, and an SDRAM external to the chip and accessible through the PL340 interface [11]. The SpiNNaker system is built with boards of 48 chips interconnected for forming a toroidal shaped triangular mesh where each chip is connected to six neighbours chips.

Sugiarto et al. [14] presented an approach for improving the overall performance of general-purpose applications running as a task graph on the same many-core neuromorphic supercomputer. Whereas in a recent paper, we have used the cortical microcircuit application as a test case for demonstrating that an enhanced partitioning and placement system studied for the SNN topology can produce a more reliable and stable configuration for the simulation on the SpiNNaker system [13].

## III. Problem Formulation

The SNN placement into the neuromorphic architecture can be view as an optimisation problem that involves two graphs: $\mathcal{G}_{\mathcal{N}}$ and $\mathcal{G}_{\text{CPU}}$.

A graph $\mathcal{G} = (V, E, \mathcal{W})$ is a mathematical representation for describing a set of elements $V$ and a set of relations $E \subseteq \{(v_i, v_j) : v_i, v_j \in V\}$ among them. The elements are called *nodes* of the graph and the relations are called *edges* of the graph. An edge $e_{ij} \in E$ binds two nodes $v_i, v_j \in V$ to each other. A graph can have a $\mathcal{W} : E \to W$ function that associates an edge $e_{ij} \in E$ to a value $w_{ij} \in W$. The value $w_{ij} = \mathcal{W}(e_{ij})$ is called edge weight. A graph can be categorised according to two properties: i) If the nodes on edges form unordered pairs $e_{ij} : \{v_i, v_j\}$ the graph is said *undirected* otherwise it is said *directed* and the nodes on edges form ordered pairs $e_{ij} : (v_i, v_j)$. ii) If the weight set $W$ is empty the graph is said *unweighted*, otherwise it is said *weighed*.

A Spiking Neural Network (SNN) can be represented using a directed and weighted graph called *neuron graph* $\mathcal{G}_{\mathcal{N}}$. In $\mathcal{G}_{\mathcal{N}}$ the nodes are the SNN neurons and the edges are the SNN synapses. Taking into account a synapse $e_{ij} : (v_i, v_j)$, the neuron $v_i$ is called pre-synaptic neuron and the neuron $v_j$ is called post-synaptic neuron. The edge weight $w_{ij}$ represent the synapse contribution to injected current into the post-synaptic neuron after a stimulus received by the pre-synaptic neuron and is called synaptic weight.

The neuromorphic architecture can be represented using an undirected and weighed graph, called *target graph* $\mathcal{G}_{\mathcal{T}}$. The graph nodes are the ARM processors, and the graph edges are the connections between them. The SpiNNaker board has 48 chip with 16 available processors each, and each chip is connected to other six chip. The edge between two processors of the same chip has a weight of one, the edge between two processors of two neighbour chip has a weight of two.

We can define the placement problem $\Pi : \mathcal{G}_{\mathcal{N}} \to \mathcal{G}_{\mathcal{T}}$ as a minimization problem (2).

$$\underset{f(\pi)}{\text{minimize}} \quad f : \sum_{e_{ij} \in E_{\mathcal{N}}} d(\pi(v_i), \pi(v_j)) \tag{1a}$$

$$\text{subject to} \quad \pi(i) = \pi(j) \to \mathcal{M}(i) = \mathcal{M}(j),\ i, j \in V_{\mathcal{N}} \tag{1b}$$

$$|\pi(i) = p| \leq \mathcal{S}(\mathcal{M}(i)),\ i \in V_{\mathcal{N}}, p \in V_{\mathcal{T}} \tag{1c}$$

The goal of a placement procedure is to minimise the *overall synaptic stretching* (2a) to reduce the communication along the network nodes. The *synaptic stretching* is the distance between the processors where two adjacent neurons are placed. Where $\pi : V_{\mathcal{N}} \to V_{\mathcal{T}}$ is the placement rule, $\mathcal{M} : V_{\mathcal{N}} \to M$ is the neuron-model association rule and, $\mathcal{S} : M \to \mathbb{N}$ is the association rule between a neuron model and the maximum number of neuron per CPU. The constraints of the placement problem are two: i) All neurons mapped into a CPU must be of the same model (2b). ii) Each CPU can simulate only a certain number of neurons, and the quantity depends on the complexity of the neuron model (2c).

### A. Problem Relaxation

A SNN is almost never described in $\mathcal{G}_{\mathcal{N}}$ form, due the high complexity in manage all neurons and synapses, but is

normally described in terms of Population and Projection. A Population $\mathcal{P}$ is a set of neurons that share the same model and the same properties. A Projection between two Population $\mathcal{P}^{(a)}$ and $\mathcal{P}^{(b)}$ defines a rule for create a set of synapses where the pre-synaptic neurons are in $\mathcal{P}^{(a)}$ and the post-synaptic neurons are in $\mathcal{P}^{(b)}$. We will refer to the *Population-Projection graph* using the notation $\mathrm{G}_{\mathcal{P}}$.

We can eliminate the two constrains (2b, 2c) redefining the problem $\Pi$ working from the graph $\mathcal{G}_{\mathcal{P}}$. The first step is splitting each population $\mathcal{P}^{(i)}$ into a set of partial populations $\left\{\mathcal{P}_1^{(i)}, \mathcal{P}_2^{(i)}, \ldots, \mathcal{P}_z^{(i)}\right\}$. All partial populations must contains at most a number of neurons equal to the maximum number of neurons allowed to be simulated in a processor: $|\mathcal{P}_j^{(i)}| \leq n^{(i)}\ \forall j = 1, \ldots, z$, with $n^{(i)} = \mathcal{S}(\mathcal{M}(\mathcal{P}^{(i)}))$.

In this way we obtain the *partial population graph* $\mathcal{G}_{\mathrm{pp}}$. The edges of the partial population graph are weighed and ordered. Given an edge $e_{ij} \in E_{\mathrm{pp}}$ between two partial population, its weight $w_{ij}$ is equal to the number of synapses shared between the neurons belonging the two partial populations.

We can redefine (2) using the placement rule $\pi : V_{\mathrm{pp}} \to V_{\mathcal{T}}$ that map a partial population into a processor (3).

$$\underset{f(\pi)}{\text{minimize}} \qquad \sum_{e_{ij} \in E_{\mathrm{pp}}} d(\pi(v_i), \pi(v_j)) * w_{ij} \tag{2a}$$

$$\text{subject to} \qquad |\pi(i) = p| \leq 1,\ i \in V_{\mathrm{pp}}, p \in V_{\mathcal{T}} \tag{2b}$$

In (3a) we modify the cost function to take into account the number of synapses shared between the processors. The rule in (3b) describes the single constraint of the problem: a processor may contain only one partial population.

### B. Graph Partitioning

The partition problem of $\mathcal{G}_{\mathcal{P}}$ can be solved in different ways. In [13] it was treated as a problem of clustering. The provided solution was divided into three step:

- Graph expansion: $\mathcal{G}_{\mathcal{P}} \to \mathcal{G}_{\mathcal{N}}$
- Spectral clustering: $\mathcal{G}_{\mathcal{N}} \to \mathbb{R}^{|V_{\mathcal{N}}|}$
- Legalization and clusters fusion: $\mathbb{R}^{|V_{\mathcal{N}}|} \to \mathcal{G}_{\mathrm{pp}}$.

The first step is to create the neuron graph $\mathcal{G}_{\mathcal{N}}$ by applying the synaptic generation rules defined into the Population-Projections graph $\mathcal{G}_{\mathcal{P}}$. In the second step, a spectral clustering procedure is applied to the neuron graph.

The Spectral Clustering involves the eigendecomposition of a representative matrix of the graph. In the case of $\mathcal{G}_{\mathcal{N}}$, a directed graph, it was used a Laplacian Matrix (4) obtained throught a transition matrix induced by a random walk [15].

$$L = I - \frac{(\Phi^{\frac{1}{2}} P \Phi^{-\frac{1}{2}} + \Phi^{-\frac{1}{2}} P^T \Phi^{\frac{1}{2}})}{2} \tag{3}$$

The results of the Spectral Clustering is the $\mathcal{G}_{\mathcal{N}}$ rapresentation into the eigenspace of $\mathbf{L}$, a space belonging to $\mathbb{R}^{|V_{\mathcal{N}}|}$. The neurons can be clustered into the eigenspace using the KMeans algorithm. After the clustering, a legalisation phase gathers in groups all neurons belonging to the same cluster and the same population. Finally, a second legalisation phase, called Fusion, builds the partial populations putting together the nearby groups of neurons until reach the maximum number of neurons that a processor can simulate.

Other techniques of graph clustering are Multilevel Graph Partitioning and Markov Cluster Algorithm [16], [17]. These techniques, like the Spectral Cluster, was born for undirected graph and their usage should be analysed using different symmetrisation techniques if applied to a directed graph.

## IV. Placement

As seen in section III our goal is placing $\mathcal{G}_{\mathcal{N}}$ into a set of processors $\mathcal{G}_{\mathcal{T}}$. In subsection III-A we have relaxed the constraints of the problem separating it into two subproblems: i) Clustering $\mathcal{G}_{\mathcal{N}}$ (or partitioning if consider $\mathcal{G}_{\mathcal{P}}$ as a starting point) into the partial population graph. ii) Placement of $\mathcal{G}_{\mathrm{pp}}$ into $\mathcal{G}_{\mathcal{T}}$. We have briefly described the clustering (or partitioning problem) in the section III-B. In this section, we independently explore the placement problem (3) by comparing different techniques: Naïve, Spectral Embedding, Scotch and Simulated Annealing.

### A. Naïve Placement

The Naïve approach is the standard mapping procedure adopted in the SpiNNaker toolchain for assigning populations of neurons to be simulated on the cores available in the SpiNNaker Platform. It is a simple and computationally light method to perform the graph placement without taking into account neither source and target graph connectivity.

The processor graph was ordered following a polar coordinate system $(\rho, \varphi)$ starting from a chip of choice. The radius $\rho = \max(|x|, |y|, |x - y|)$ has been calculated using the hexagonal distance. The angle $\varphi \in [0, 2\pi)$ is expressed in radians. The procedure starts to place a partial population into each processor and change the chip when all processors inside a chip are used. As the $\rho$ increases, the sub-populations will be distributed along the chip on the circumference and will be separated by a greater and greater distance.

### B. Spectral Embedding

The Spectral Embedding placement was partially used in a previous work described in [13]. The procedure involves the spectral analysis of the graph and a dimension reduction procedure to obtain a planar representation of it. By doing so, the target graph can be directly superimposed on the graph of the partial populations. Contrary to previous work, in which a greedy heuristic was used, the association of partial populations with processors was finally described through an *Integer Linear Programming* (ILP) problem.

The procedure starts with the extraction of the first five eigenvalues, and the relative eigenvectors, from the matrix $\mathbf{L}$. The eigenvectors form a matrix $\mathbf{\Lambda}$ that represents the partial populations in a $\mathbb{R}^5$ space. We apply a non-linear dimension reduction procedure using *Sammon Mapping* obtaining a space in $\mathbb{R}^2$.

The Sammon Mapping algorithm minimise the error function in (5) where $d_{ij}$ is the distance in the high-dimensional space (eigenspace) and $d_{ij}^*$ is the distance in the low-dimensional space (placement space) [18].

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

Each chip, in the chip mesh, is represented as a point $(x, y)$ in an axial coordinate system. We superimpose the graph $\mathcal{G}_\mathcal{T}$ on $\mathcal{G}_{\text{pp}}$ projecting the chip mesh in the placement space (7).

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \sqrt{\frac{2A_h}{3\sqrt{3}}} \begin{pmatrix} \sqrt{3} & -\frac{\sqrt{3}}{2} \\ 0 & \frac{3}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{5}$$

Where $(x, y)$ is the chip coordinate in the hex mesh, and $(x^*, y^*)$ is the chip coordinate in the placement space. The side length of the hex is used as a normalising factor and calculated using the area $A_h = \frac{A}{m}$ occupied by each chip. The normalising factor allows scaling the chip mesh concerning the area $A$ occupied by the partial populations. After projecting the points into the placement space, they are translated to centre them on the median of the points representing the partial populations. Now we can describe the placement problem using the ILP formulation (8).

$$\underset{f(\mathbf{X})}{\text{minimize}} \qquad f : \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} d_{i,j} \tag{6a}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} x_{i,j} \leq k \quad \forall j \in \{1, \ldots, m\} \tag{6b}$$

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad \forall i \in \{1, \ldots, n\} \tag{6c}$$

Where the $\mathbf{X} = (x_{ij})$, $x_{ij} \in \{0, 1\}$ matrix is the placement matrix. An entry $x_{ij} = 1$ means that partial population $i$ is mapped on the target node $j$. The problem constraints are two: i) Each target node can host at most $k$ partial populations (8b). ii) Each partial population can be associated to only one target node (8c). The ILP problem was modelled using PuLP Python library and solved with *COIN-OR branch and cut* (CBC) solver.

### C. Scotch

The Scotch mapping procedure makes use of the programs available in the homonym software suite (SCOTCH). The Dual Recursive Bipartitioning (DRB) is the primary procedure used by this tool [19]. The DRB can use a plethora of other bi-partitioning methods according to a strategy defined by the user or deducted by graph properties. The main available methods are: Gibbs-Poole-Stockmeyer [20], Fiduccia-Mattheyses [21], Greedy Graph Growing [16] and Diffusion [22].

The mapping workflow with SCOTCH plans to pre-partition the target graph through the *amk_grf* program. The *amk_grf* program take in input a graph in *grf* format and create a target file (*tgt* format) which contains a decomposition-defined target architecture of same topology as the input graph.

Once a decomposition of the target graph has been obtained, the graph of the partial populations is placed on the target graph using the *gmap* program. The program *gmap* take in input the partial population graph in *grf* format and the target graph in *tgt* format and perform the DRB procedure minimising the communication cost function[1]. The *gmap* output file is a mapping file (*map* format) that contains the association between the Source and the Target nodes.

[1]The SCOTCH cost function is similar to our Synaptic Stretching

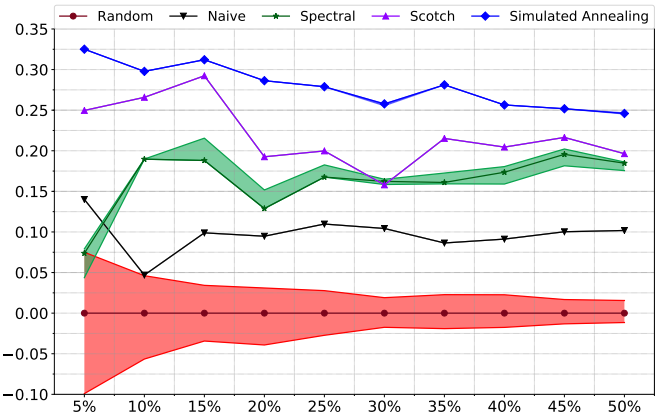



Fig. 1: The graph represents the improvement of a mapping technique with respect to the median of the results obtained with a random placement, y-axis. The x-axis shows the CM scale factor.

We had developed a Python module able to exporting a NetworkX graph to a file according to the *grf* format used by SCOTCH and capable of automating the procedures described above.

### D. Simulated Annealing

The Simulated Annealing is a well know procedure used to find a good solution to an optimisation problem [23]. Given the problem in (3a), it is convenient to express the overall synaptic stretching in a matrix form and define a cost function to minimise. Given the partial population graph $\mathcal{G}_{\text{pp}}$ we build its Adjacency matrix $\mathbf{A} = (a_{ij})$ as described in (9).

$$a_{ij} = \begin{cases} w_{ij} & \text{if } \exists (v_i, v_j) \in E_{\text{pp}} \\ 0 & \text{otherwise} \end{cases} \quad \forall i, j \in \{1, \ldots, n\} \tag{7}$$

Given the processor graph $\mathcal{G}_\mathcal{T}$ we build its distance matrix $\mathbf{D} = (d_{ij})$ where each entry $d_{ij}$ is the lenght of the mimimum path between two target nodes $\text{cpu}_i$ and $\text{cpu}_j$. The distance matrix can be build using the Floyd–Warshall algorithms or repeating Dijkstra's algorithms if $|E_\mathcal{T}| \ll |V_\mathcal{T}|^2$.

Assuming to have as many subpopulations as processors and a placement rule $\Pi : \{v_1, \ldots, v_n\} \to \{\text{cpu}_1, \ldots, \text{cpu}_n\}$ we construct the *permutation vector* $\pi : (\Pi(v_1), \ldots, \Pi(v_n))$ and the *permutation matrix* $\mathbf{P}_\pi = (p_{ij})$ in row form (10).

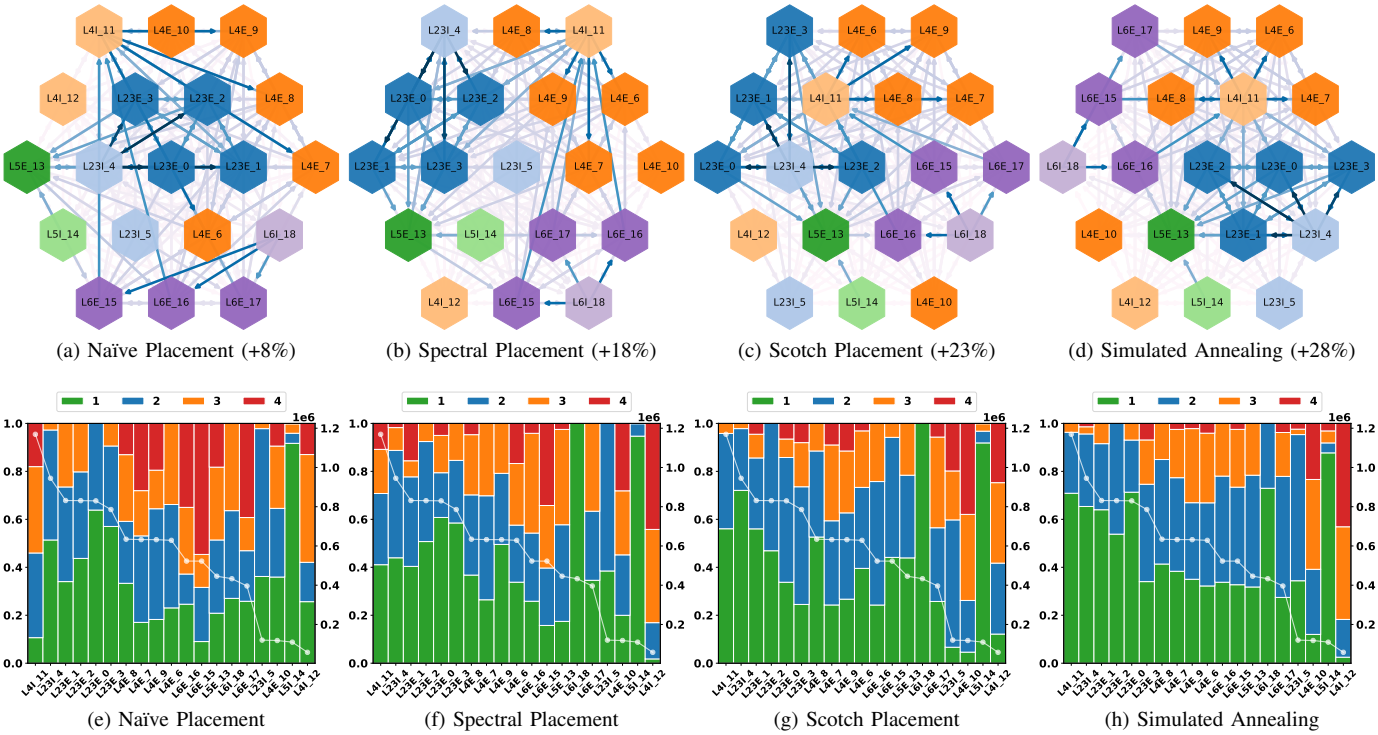
$$p_{ij} = \begin{cases} 1 & \text{if } i = \pi_j \\ 0 & \text{otherwise} \end{cases} \quad \forall i, j \in \{1, \ldots, n\} \tag{8}$$

The permutation matrix is applied to $\mathbf{D}$ to permutate its rows and columns. We obtain the matrix $\mathbf{D}_\pi = \mathbf{P}_\pi \mathbf{D} \mathbf{P}_\pi$. The overall synaptic stretching can be expressed in a matrix form and used as the cost function for the simulated annealing algorithm (11).

$$f : \mathbf{e}^T (\mathbf{A} \odot \mathbf{D}_\pi) \mathbf{e} = \sum_{i,j} a_{ij} * d^{(\pi)}_{ij} \tag{9}$$

Where $\odot$ is an element-wise multiplication and $\mathbf{e}$ is a column vector whose all elements are equal to one.

We used the Simulated Annealing implementation provided in the SciPy ecosystem using the *temperature* to decide how many elements of the permutation vector $\pi$ to swap.



(a) Naïve Placement (+8%) (b) Spectral Placement (+18%) (c) Scotch Placement (+23%) (d) Simulated Annealing (+28%)

(e) Naïve Placement (f) Spectral Placement (g) Scotch Placement (h) Simulated Annealing

Fig. 2: The figures in the first row represent the placement of the partial population graph build from a $\text{CM}_{20\%}$ with 1000 neurons per chip on 19 chip (5 processors per chip). The figures in the second-row represent for each partial population the number of synapses (white line) and the percentage of synapse stretching.

## V. Results

In this section, we present the exploration experiments using the methods described in Section IV.

We use the Cortical Microcircuit (CM) as benchmark network, [8]. This network represents the connectivity of neurons inside a slice of the cerebral cortex with an area of $1\,mm^2$. The CM has been chosen because it is a rapresentative biological model with a relativly high global connectivity (5%) and natural clusters defined by the four cerebral cortex layers $\{L_{23}, L_4, L_5, L_6\}$. The CM is described in terms of Population and Projection with two populations for each layer, for a total of 8 Population and 64 Projections.

The network is composed of Integrate and Fire (LIF) and Spike Source (SRC) neuron models. The LIF neurons are models that mimic the biological neurons behaviour. The SRC neurons are simple programmable applications for outputting signals when desired. In this network, the SRC neurons are used to simulate the background activity of cortical neurons not presents in the model. Each SRC neuron is connected to only one LIF neuron, so they can be excluded by the $\mathcal{G}_{\text{N}}$ provided that processors are reserved for their execution.

The CM model has 7.72e+4 LIF neurons and 2.99e+8 synapses. The network can be down-scaled to a percentage $\text{CM}_p$, for example:

- $\text{CM}_{5\%}$ has 3.86e+3 neurons and 7.47e+5 synapses.
- $\text{CM}_{10\%}$ has 7.72e+3 neurons and 2.99e+6 synapses.
- $\text{CM}_{50\%}$ has 3.86e+4 neurons and 7.47e+7 synapses.

For each processor in charge of simulating a LIF partial population, we must reserve two further processors. A processor is reserved for the simulation of paired SRC neurons. A further processor is reserved to host a special application necessary to manage synapses with delays greater than 10 ms, as described in [24]. Taking into account a set of 16 processors belonging to the same chip, we can place 5 partial population per chip for a total of a thousand neurons per chip.

For simplifying the problem we perform a sequential slicing of each population in order to obtain partial populations with at most 1 000 neurons. In this way, we use an entire chip (5 processors with 200 LIF neurons each) for a partial population.

The experiment environment is composed of four different mapping procedures: Naïve, Spectral, Scotch and Simulated Annealing. We had generated 5 CM networks for 10 different scale factors, from 5% to 50%, for a total of 50 networks. For each network, we applied all mapping procedures 20 times. We evaluate the performance of each mapping procedure for each scale factor, using the fitness function (11). As a result, we obtain a distribution of 100 different placement results concerning overall synaptic stretching.

The performance of mapping procedures is compared to the performance of random placement. The median value of the results obtained with the Random procedure is used to compute the percentage improvement of the results obtained with other techniques.

In figure 3 is depicted a chart that summarize all the experiments. On the x-axis, there are the network scale factors, on the y-axis the percentage placement improvements versus random. The data series are represented by polylines of different colours representing the medians of the results set. Each polyline is drawn within an area whose extremes delimit the first and third quartile of the results set.

In figure 4 are depicted the mapping results of a $CM_{20\%}$ into a target graph of 19 chip using the four placement techniques. Each hex represents a SpiNNaker chip connected with six neighbours. The colour of the hex area points out the belonging of the neurons, mapped on the chip, to one of the eight populations of the CM. The number of synapses shared between two partial populations is highlighted with the colour intensity of the edge that connects them. The different concentration of the connections with more synapses can be appreciated qualitatively from the figures 4a to 4d and quantitatively from the figures 4e to 4h.

In figure 4a can be seen how the Näive method does not consider the connectivity but place each partial population sequentially following the polar ordering of the chip. Indeed there are many connections with a large number of synapses directed towards distant chips. This not happens in figure 4d where the Simulated Annealing can localise in a defined area all partial population with a high number of shared synapses. In figures 4e and 4h the same information can be appreciated quantitatively. The chart has a bar for each partial population. Each bar represents the overall outgoing synapses of a partial population and shows the percentage of synapses at different levels of elongation. The white line depicts the number of synapses belonging to each partial population. The partial populations are sorted in descending order according to the total number of synapses.

We can see how better methods improve the percentage of synapses at a distance of 1 chip (Green) and decrease the percentage of synapses at a distance of 4 chips (Red).

## VI. Conclusions

In this paper, we described a mapping problem that involves a complex directed graph to be placed in a mesh of processors. We have modelled the mapping problem of SNN into SpiNNaker processor-mesh and split the problem into 3 phases: the expansion, clustering, and mapping. Focusing on the mapping phase, we have identified and test 4 methodologies to solve the problem. The *Naïve* method maintains the proximity of clusters but does not take into account their connectivity. The *Spectral* method uses the graph eigendecomposition to obtain a planar representation of it and perform the node association with the chip mesh through an ILP formulation. The *Scotch* method uses the Dual Recursive Bipartitioning heuristic for fast mapping of a source graph into a target graph. The *Simulated Annealing* method uses the well-known procedure to minimise a cost function.

We are redefining the cost function of the placement problem bringing it into matrix form as a function of a permutation vector. We have chosen the cortical microcircuit at different scale factors as our benchmark network, preferring it for its high connectivity and the presence of clusters. After performing several tests on the chosen benchmark network, the results highlight the superiority of the Simulated Annealing method that works natively on direct graphs. This modelling system for SNN placement problems can be adapted to other architectures such as Intel Lohi and SpiNNaker 2 for investigating new mapping techniques to be adopted for improving the usability of these emerging architectures.

In the next works, we will implement these techniques within the placement pipeline of the SpiNNaker neuromorphic architecture, to offer an alternative to the currently implemented method (Naïve) and evaluating experimentally the reduction of communications between the chips involved.

## Acknowledgment

The research leading to these results has received funding from European Union Horizon 2020 Programme [H2020/2014-20] under grant agreements no. 720270 and no. 785907 [HBP].

## References

[1] M. Ruggiero, A. Guerri, D. Bertozzi, M. Milano, and L. Benini, "A fast and accurate technique for mapping parallel applications on stream-oriented mpsoc platforms with communication awareness," *International Journal of Parallel Programming*, vol. 36, no. 1, pp. 3–36, 2008.

[2] Z. Á. Mann, "Multicore-aware virtual machine placement in cloud data centers," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3357–3369, 2016.

[3] 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.

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

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

[6] 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.

[7] S. J. Van Albada, A. G. Rowley, M. Hopkins, M. Schmidt, J. Senk, A. B. Stokes, F. Galluppi, D. R. Lester, M. Diesmann, and S. B. Furber, "Full-scale simulation of a cortical microcircuit on spinnaker," in *Front. Neuroinform. Conference Abstract: Neuroinformatics*, vol. 10, 2016.

[8] 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.

[9] 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.

[10] X. Jin, S. Furber, and J. Woods, "Efficient modelling of spiking neural networks on a scalable chip multiprocessor," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, June 2008, pp. 2812–2819.

[11] 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.

[12] X. Jin, F. Galluppi, C. Patterson, A. Rast, S. Davies, S. Temple, and S. Furber, "Algorithm and software for simulation of spiking neural networks on the multi-chip spinnaker system," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[13] G. Urgese, F. Barchi, E. Macii, and A. Acquaviva, "Optimizing network traffic for spiking neural network simulations on densely interconnected many-core neuromorphic platforms," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[14] I. Sugiarto, P. Campos, N. Dahir, G. Tempesti, and S. Furber, "Optimized task graph mapping on a many-core neuromorphic supercomputer," in *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*. IEEE, 2017, pp. 1–7.

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

[16] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[17] S. Van Dongen, "Graph clustering via a discrete uncoupling process," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 121–141, 2008.

[18] J. W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Transactions on computers*, vol. 100, no. 5, pp. 401–409, 1969.

[19] F. Pellegrini, "Static mapping by dual recursive bipartitioning of process architecture graphs," in *Scalable High-Performance Computing Conference, 1994., Proceedings of the*. IEEE, 1994, pp. 486–493.

[20] N. E. Gibbs, W. G. Poole Jr, and P. K. Stockmeyer, "A comparison of several bandwidth and profile reduction algorithms," *ACM Transactions on Mathematical Software (TOMS)*, vol. 2, no. 4, pp. 322–330, 1976.

[21] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Papers on Twenty-five years of electronic design automation*. ACM, 1988, pp. 241–247.

[22] F. Pellegrini, "A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries," in *European Conference on Parallel Processing*. Springer, 2007, pp. 195–204.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[24] 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.