Population-Based Routing in the SpiNNaker Neuromorphic Architecture

Sergio Davies, Javier Navaridas, Francesco Galluppi and Steve Furber
School of Computer Science
The University of Manchester
Manchester - UK
Email: daviess@cs.man.ac.uk, javier.navaridas@manchester.ac.uk, galluppf@cs.man.ac.uk, steve.furber@manchester.ac.uk

***Abstract*—SpiNNaker is a hardware-based massively-parallel real-time universal neural network simulator designed to simulate large-scale spiking neural networks. Spikes are distributed across the system using a multicast packet router. Each packet represents an event (spike) generated by a neuron. On the basis of the source of the spike (chip, core and neuron), the routers distribute the network packet across the system towards the destination neuron(s). This paper describes a novel approach to the projection routing problem that shows advantages in both the size of the routing tables generated and the computational complexity for the generation of routing tables. To achieve this, spikes are routed on the basis of the source population, leaving to the destination core the duty to propagate the received spike to the appropriate neuron(s).**

I. Introduction

Biological neurons communicate in networks with spectacular levels of connectivity using signals that are remarkably simple. The signals used for this purpose are electrical pulses known as action potentials or spikes. The information transmitted from a neuron to subsequent ones in the network is encoded in trains or sequences of spikes. Spikes are transmitted from neuron to neuron more-or-less losslessly over axons which distribute the action potential to many targets.

The structure of the input dendritic tree allows a neuron to receive inputs from many thousands of other neurons through synaptic connections. The nature of the network – simple processing, complex connectivity – suggests that much of the remarkable processing power of the brain must stem from the connection topology.

The need to support efficient processing within the space and wiring constraints of the brain naturally suggests some internal structure: biological neural networks do not have random topologies. While specific functional microcircuits within biological neural networks have been identified only for certain cases (e.g. [1]), the global connectivity of the brain is known to feature bundles of long-range connections connecting specific regions to other specific regions in tight clusters [2]. Likewise the general structure of the cortex has been analysed and is understood to have dense local connectivity with relatively sparse, but strongly-directed, long-range connectivity [3]. The overall pattern – heavily clustered populations of neurons communicating through narrow specific projections – is thought to provide a generic modular architecture to the brain that permits efficient processing of almost any function or behaviour within a universal substrate.

Nonetheless it seems remarkable that attempts to create neural networks in hardware have rarely emphasised or even implemented this type of architecture. First-generation neural processors, on the whole, tended to implement either cellular connectivity or all-to-all patterns that promote reduced locality [4]. A second generation, acknowledging the need for reconfigurablity, adopted a rewirable architecture, but on the whole tended towards the circuit-switched approach typical of FPGAs that is unsuitable for biological-style topologies [5].

Imitating the spiking behaviour of biological neurons, a third generation of computational artificial neural networks has been proposed [6]. Simulators for third generation artificial neural networks have been developed using various technologies and techniques [7] [8]. Software simulators require the implementation of a computer program that solves the mathematical models on usually unsuitable, inefficient sequential digital computers. Hardware simulators have thus also emerged promising more efficient computation [9]. Such chips may use an Address Event Representation protocol (AER) to distribute the event across the system (in the case of a digital communication system), or a matrix of analogue components that emulate the required inter-neuron dynamics (in the case of purely analogue hardware). Some examples of neuron intercommunication paradigms, as implemented in hardware spiking neural network simulators [8], are:

- **Fully connected networks**: neurons communicate with all or a subset of the remaining neurons;
- **Locally connected networks**: neurons are arranged in a matrix in which each neuron communicates only with a subset of the neighbouring neurons;
- **Layered architectures**: neurons of a specific layer communicate only with neurons of the layers up-stream and down-stream;
- **Reconfigurable architectures**: the connections of each neuron are reconfigurable both in synaptic destination and in synaptic weight.

A common feature of all these architectures is that only *neuron-to-neuron* connections are taken into account, i.e. the description of the connectivity always uses a single neuron as source and a single neuron as destination. However, in biology, it is known that neurons are functionally aggregated into populations (called "cell assemblies" in [10]) where all the neurons contribute to the same function, and then populations are interconnected [11]. Indeed, in the biology the per-unit computation seems to be very simple while the sophistication is in the connectivity. It is surprising, therefore, that during over 2 decades of hardware (and software) development, the effort has focussed more on efficient *computational methods* rather than on efficient connectivity architectures. This idea seems always to have been latent in the field [12], but never exploited.

In this paper we present an architecture based on a routable AER network that permits a more biologically-similar connectivity and a software approach to configure it, based on the novel idea of connecting *populations*. Hence all the neurons in a population share a common path to the destination population(s). The precise description of the connection between each single neuron happens only at the very last operation, when the spike has to be distributed within the destination population that resides in a specific simulation unit. The population-based approach is then tested with various models of neural networks and shown to be an efficient approach to the neuron intercommunication problem.

II. Introduction to SpiNNaker

SpiNNaker [13] is a hardware-based real-time universal neural network simulator following an event-driven computational approach [14]. This project involves the design of a chip and the development of dedicated software to simulate neural networks [15]. This system tries to mimic the features of biological neural networks in various ways:

- **Native parallelism**: each neuron is a primitive computational element within a massively parallel system. Likewise, SpiNNaker uses parallel computation;
- **Spiking communications**: in biology, neurons communicate through spikes. The SpiNNaker architecture uses source-based AER packets to transmit the equivalent of neural signals;
- **Event-driven behaviour**: neurons are very power efficient, and consume much less power than modern hardware. To reduce power consumption we put the hardware in "sleep" state waiting for an event [15];
- **Distributed memory**: in biology, neurons use only local information to process incoming stimuli. In the SpiNNaker architecture we use memory local to each of the cores and an SDRAM local to each chip;
- **Reconfigurability**: in biology, synapses are plastic. This means that the neural connectivity change in both shape and strength. The SpiNNaker architecture allows on-the-fly reconfiguration.

Here we introduce the main features of the SpiNNaker system. Complete details of the system and of its performance can be found in [13], [14], [16], [17], [18] and [19].

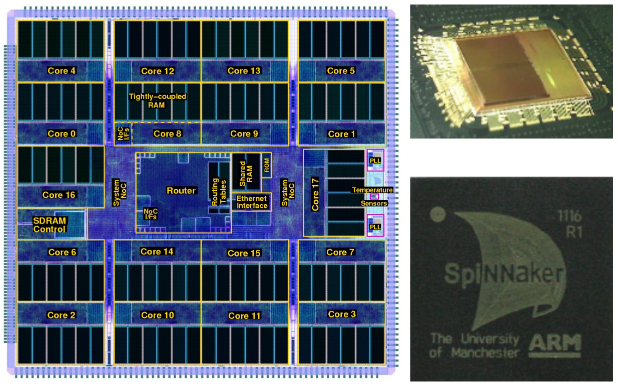


Fig. 1. SpiNNaker chip layout with labels for functional blocks and pictures of the chip before and after packaging. Image taken from the GDS2 plot sent to the manufacturer. The chip was fabricated at UMC using their 130e-llsp low-power process. Die size is 10×10 mm.

A. Hardware

alghero airport The core of the SpiNNaker system is the SpiNNaker node (fig.1): a multiprocessor chip including 18 ARM968 processors (see fig. 2), each running at 200 MHz. Each core has a 96 KB TCM (Tightly Coupled Memory) containing local instructions and data. In addition, a shared 128 MB SDRAM memory is available to all the cores of a chip.

A network router [17] provides interconnection features to the cores internal and external to the chip. While the ARM IP blocks are off-the-shelf general purpose programmable processors, the router is a completely custom component optimized for spiking models (although its use is not limited to spiking neural network simulation).

SpiNNaker chips integrate into a large-scale system normally consisting of a regular two-dimensional hexagonal array of up to 256×256 chips. The whole network is then wrapped to form a toroidal shape (fig. 3). Some of the nodes in this network are connected to the external world by Ethernet links, through which it is possible to send data, code, and commands to the SpiNNaker system. It is worth noting in particular the programmability of this system: although it has been designed to perform with spiking neural network simulations, the general purpose cores embedded in the SpiNNaker chip make this hardware versatile and usable also in other contexts and for different purposes.

B. Software

The software developed for this system has been designed to maximise power efficiency. When idle, the processor is kept in low-power sleep mode with interrupts enabled. When an interrupt is received, the processor performs the required actions to respond to the interrupt request and then returns to sleep. Details of the software architecture have been described in various publications: [15] [20] [21]. It is important to note that the number of neurons that each core is able to simulate within the real-time constraint varies with the computational complexity of the neuron model itself and with

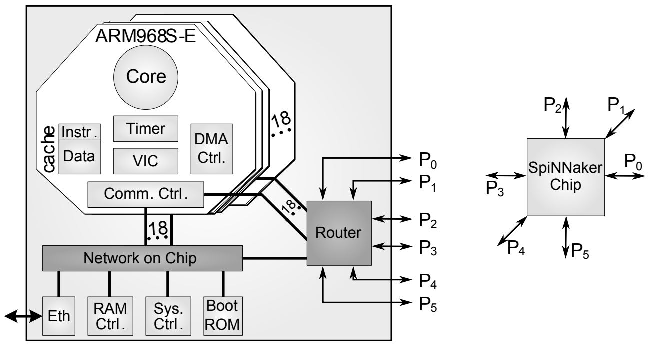


Fig. 2. Schematic of the SpiNNaker chip with the view of the routing directions.

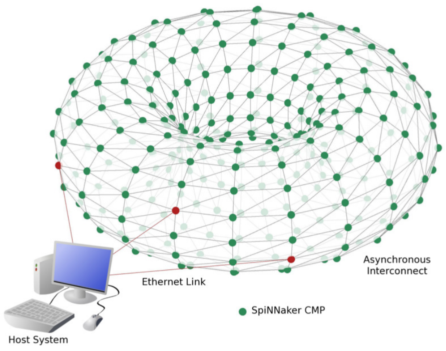


Fig. 3. The SpiNNaker system: each dot is a SpiNNaker chip. The lines between them are the links. The system is wrapped in a toroidal configuration.

the connectivity pattern required [22]. However, the maximum number of neurons that each core is able to simulate is imposed by the design specifications of the SpiNNaker system. All the components, in particular those related to memory access and communication, have been designed to support the traffic generated by at most $1,000$ neurons per core.

C. Multicast communications in SpiNNaker

As described above, when a biological neuron emits an action potential, it influences a large number of post-synaptic neurons, typically in the order of thousands. This mechanism has been reproduced in the SpiNNaker system using a multicast packet router that allows on-chip and inter-chip communications. Its main task is to propagate network packets to destination cores on the basis of the identifier of the source neuron.

When a neuron fires, a packet is generated and transmitted through the network. The only information contained in the packet is the identifier of the neuron that has fired (see fig.4). The information necessary to deliver the packet to the appropriate destinations is embedded in the routing table present in each router [17]. When a router receives a packet, it looks in the routing table for a match to determine the direction of the following step. Each routing entry is formed by three registers: key, mask and direction. The key field identifies the routing key to match for the entry to be selected. The mask

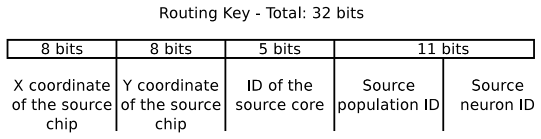

Routing Key - Total: 32 bits
8 bits
8 bits
5 bits
11 bits
X coordinate of the source chip
Y coordinate of the source chip
ID of the source core
Source population ID
Source neuron ID

Fig. 4. Description of the routing key structure.

field identifies the bits of the key that have to match with the routing key of the packet received for the entry to be selected. If a masked entry matches the received routing key, then the corresponding direction register contains the direction towards which the packet is propagated. If no entries match the received routing key, a mechanism for "default routing" is triggered and the packet is transmitted to the link opposite to the one from which it has been received (e.g. a packet received from link 3 is default-routed to link 0 in fig.2).

Given that the high fan-out of neural connectivity, generating the distribution trees and filling these routing tables is a very important, *non-trivial* part of the neural computation: in this paper we explore a new approach that simplifies this task. Instead of generating neuron-to-neuron multicast trees we greatly reduce computation time by generating population-to-population routes.

In the following sections of the paper the term "hop" is used to describe the passage of a projection (or, equivalently, a packet) through one router of a SpiNNaker chip.

III. POPULATION-BASED ROUTING

In the SpiNNaker system, each action potential generated by a neuron is sent to the destination core(s) using a multicast packet on the basis of the source neuron identifier. The routing key comprises 32 bits that uniquely identify the source neuron that emitted it.

The 32 bits forming a routing key can be used arbitrarily, since the routers and the cores are fully programmable. However, for the sake of simplicity, we defined a convenient structure with four fields described in fig.4. The first three fields (21 bits) provide information about where the neuron is placed. The fourth field is further split into two parts: a *population* identifier and a *neuron* identifier. These two sub-fields have varying sizes so that it is possible to adapt them to the size of the populations simulated in each core. If populations are big, there can be fewer populations inside a core (few bits for the population identifier, many bits for the neuron identifier). On the other hand, inside each core there can be a greater number of smaller populations (many bits for the population identifier, few bits for the neuron identifier). If populations exceed the number of neurons that a core is able to simulate, they have to be split in appropriately sized parts which will be treated as individual populations, returning the problem to the original population-based routing.

The masking capability implemented in the router allows us to select the bits on the basis of which the packet has to be routed. Therefore we took advantage of this feature using a mask that selects the part of the routing key related to the first four fields depicted in fig.4, which represents the complete source population identifier.

For the purpose of the generation of the routing key, the populations are sorted inside each core by their size. In addition, the size of the population is rounded up to the nearest power of two. This addressing scheme has the advantage of subdividing the routing key so that all neurons belonging to the same population have the same initial pattern, which is the population key identifier code (population ID). However there is the disadvantage that some of the routing keys are not actually used for specific neurons, since they are used to align the following routing keys so that each population has its own identifier inside a core.

The greatest number of unused routing keys is generated if a population contains $2^n + 1$ neurons: in fact the size of the neuron identifier is then $n+1$ bits, (2^{n+1} possible routing keys) of which 2^n+1 are actually used and 2^n-1 are unused, with a ratio of $\approx 50\%$. Therefore, to include this worst case scenario in the addressing scheme, we assign 11 bits of the routing key to the population and neuron identifier. In total this addressing space is able to identify $2,048$ neurons per core in the SpiNNaker system, giving twice the addressing space strictly required for the simulation.

However, the key point of the approach to the population-based routing is the definition of the population key. In the following two examples we do not focus on the first part of the routing key which includes the X and Y coordinates of the chip and the core identifier, but focus only on the section related to the population ID and neuron ID, describing how these values are computed. In a simple example, if a core contains a single population of 60 neurons, the population will be assigned the routing keys in the set 00000000000 to 00000111111, even though the last four routing keys will not be used. Therefore the population ID assigned to this population comprises 5 bits 00000, which is the immutable part of the set of the assigned routing keys:

$$\overbrace{00000}^{\text{Population ID}}\quad\overbrace{xxxxxx}^{\text{Neuron IDs}}$$

In a more complex example, if a core contains three populations of (A) 60, (B) 20 and (C) 6 neurons, the routing keys will be assigned consecutively:

Pop. Name	Pop. size	Routing key start	Routing key end	Pop. ID	Neuron IDs
A	60	00000000000	00000111111	00000	xxxxxx
B	20	00001000000	00001011111	000010	xxxxx
C	6	00001100000	00001100111	00001100	xxx

For the purpose of routing the packets in the SpiNNaker network, it is possible to use a single entry in the router per population, defining the route on the basis of the population ID. Since this has a variable size, the mask field of each routing entry must be computed according to the population size (rounded up to the greater power of two). In this way, even if populations are formed by a number of neurons which is not exactly a power of two, we can still use a single entry per population to route packets.

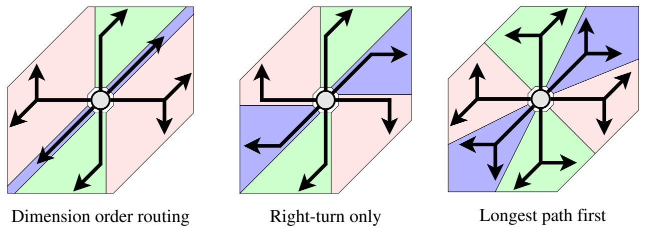


Fig. 5. Shape of the routes generated with each algorithm.

A. Multicast route generation

Given that the generation of the routes is outside of the scope of this paper, we will consider only simple algorithms for generating the multicast trees. All the algorithms discussed in this section are well-known in the literature [23] and generate routes obliviously, *i.e.* they only require local information. This is a prerequisite for any multicast generation algorithm that has to be used in SpiNNaker, because for large configurations of the machine each chip will be in charge of generating the multicast trees for its hosted neurons independently of the rest of the system. Furthermore, we will consider only the shortest path from a source to each destination, which, given the triangular topology, will require, at most, advancing in two of the three possible directions (X, Y, Diag). We will consider the following algorithms:

- Dimension Order Routing (DOR): This routes the packets first advancing all the required hops in the X dimension, then the Y dimension and finally along the diagonal.
- Right Turn Only (RTO): Packets are routed in a way such that only right-turns are allowed. This require prioritizing the use of directions clockwise cyclically.
- Longest Path First (LPF): Each packet advances first through the direction having the largest number of hops and then in the other one (if any).

Fig. 5 shows the shape of the routes generated by each algorithm and the areas covered by each local output port. In the figure it is apparent that LPF has a better partitioning of the network and a more balanced use of network resources. To corroborate this, we have performed a simple experiment. We have created $1,000$ random collections of 256 destination nodes located, on average, 32 hops from the source. Then, using different algorithms, we have generated the multicast trees and measured the network resources used by each algorithm. The results of this experiment are plotted in fig. 6. It shows the average resources used by each algorithm as well as the resource used per dimension. The latter provides an insight into the balance of the use of network resources. Note that an unbalanced use of network resources may lead the network to suboptimal behaviour. The difference in terms of entries in the routing tables used by the different algorithms is

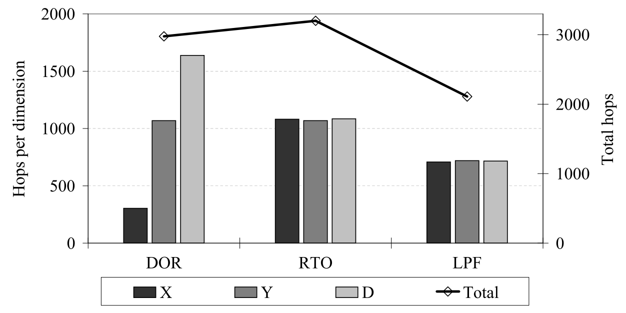


Fig. 6. Network resources needed by each algorithm.

negligible (less than 0.1%) and therefore will not be considered for deciding among them.

The results show that DOR has a very unbalanced use of network resources. It requires a greater use of the diagonal links (roughly 1.5× Y and 5× X) which are therefore likely to become a bottleneck. RTO provides a much more balanced use of the network, but at the cost of increasing the total network resources employed. However, as its use per dimension is noticeably lower than the most used dimension in DOR, it does compensate for the higher use of resources in other dimensions. Finally, LPF has the lowest requirements in terms of network resources (roughly $\frac{2}{3}$ of the other algorithms) and also produces a very balanced multicast tree. For this reason, we selected LPF to be implemented in SpiNNaker as a first approach to generate multicast routes and we use this algorithm throughout the rest of this paper.

B. Implementing LPF for the SpiNNaker system

The first step to compute the routing path is to evaluate the difference between the coordinates of the source and the destination chip. Doing so, we have a number of hops in the four main directions (directions 0, 2, 3 and 5 as described in fig.2).

The second step is to compute the number of hops in the diagonal direction. The number of hops in this direction will be different from 0 only if the number of hops in the X direction and the Y direction have the same sign (both positive or both negative). If diagonal hops are introduced, then the number of hops in the X and Y directions are re-computed accordingly. An example of the routing trees is described in fig.7. The figure shows a 7 × 7 subnetwork with the beginning of the branches generated by this algorithm. Squares represent nodes, and lines between them represent the links. The links that may be used by the multicast algorithm are darkened. The domains reachable from each output port are delimited by the red dashed line.

The routing paths for each of the source populations are computed step by step: each routing key is stored in a memory structure that reproduces the 256 × 256 structure of the SpiNNaker system, and then the routing algorithm moves the packet forward by one step, storing the routing key in the appropriate memory structure. Finally all these (raw) entries are further processed so that if multiple entries belonging to

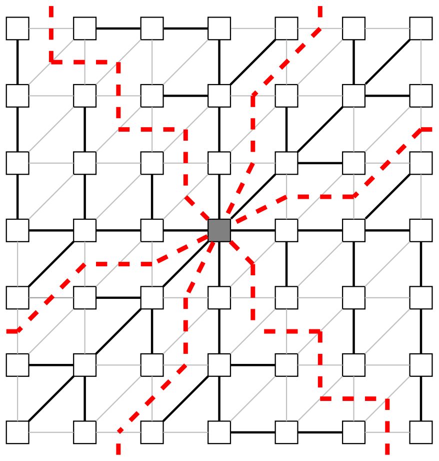


Fig. 7. Links selectable by the "Longest Path First" algorithm.

the same routing keys are present in the memory structure representing one router, they are compressed into a single entry with multiple destinations. At the same time, the default routing path is taken into account so that if an entry has to be default-routed, the entry will not appear in the final list of routing entries.

In addition, to avoid excessive memory consumption in the host during the computation of the routes, a "tiling" method has been applied: the whole 256 × 256 SpiNNaker chip assembly is divided into 64 blocks of 32 × 32 chips which are routed at once. The routing algorithm is applied to the connections which are sourced in each of the blocks of chips and then the resulting entries are compressed in the final list, freeing the memory for the next iteration.

IV. Routing tests

The implementation of our novel population-to-population routing algorithm has been tested in three different scenarios:

1) One population per core, local projections only;
2) One population per core, system-wide connectivity;
3) A simple biologically-inspired thalamocortical model.

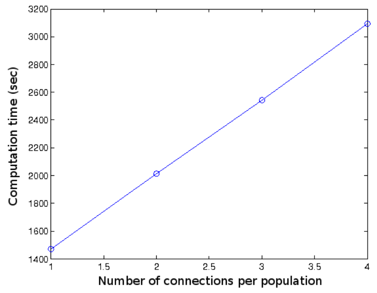
Local projections are projections between populations of neurons allocated to the same chip. Long-range projections are projections between populations of neurons that do not reside in the same chip.

In the first two tests, the generated neural network consists of one population of 512 neurons for each of the cores in the complete SpiNNaker system: 16 cores are used for neural simulation in each of the 256 × 256 SpiNNaker chips of the whole system, for a total of $1,048,576$ populations and $536,870,912$ neurons.

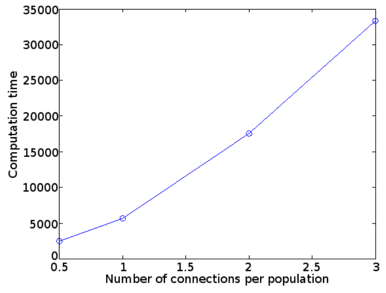
Results have been obtained running the routing algorithm on an Intel Core 2 Duo T9600 (2.8 GHz) with 4 Gb of RAM using a Python program.

A. One population per core, local projections only

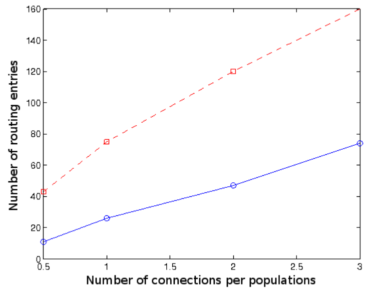
The first test involved the generation of the routing tables for a neural network where only projections within the same chip



(a) Results of the first test with local projections only. The graph presents the relation between the number of connections per source population and the time required for the computation (in seconds).



(b) Results of the second test with long-range projections only. The graph presents the relation between the number of connections per source population and the time required for the computation (in seconds).



(c) Results of the second test with long-range projections only. The graph presents the relation between the number of connections per source population and the number of entries in the routing tables. The blue solid line with circles represents the minimum number of routing entries. The red dashed line with squares represents the maximum number of routing entries.

Fig. 8. Results produced using the population-based routing principles.

were present. These projections involve, therefore, a single hop through the local chip router to reach the destination core. In fig.8(a) the time required for the computation is presented. On the horizontal axis is the number of connections per source population; on the vertical axis is the time required for the computation (in seconds). It is important to note that the time required is linear with the number of connections defined.

B. *One population per core, system-wide connectivity*

Our second test involved the generation of routes for a complete SpiNNaker system. Every population selects randomly a collection of p destination populations located all over the system; note that self-connections and local connections are possible. In this case we are interested both in the time required for the computation and the number of routing entries generated, therefore two graphs are presented: fig.8(b) shows the computational time required to route the corresponding number of projections. Fig.8(c) shows the maximum and the minimum number of entries occupied in the routing tables. In both graphs, the horizontal axis represents the number of connections per source populations (p). In the case of 0.5 connections per source populations, only half of the populations in a chip have a projection towards another population in the system.

In fig.8(b) we can see that the computational time is no longer linear with the number of connections. Since we have already shown that the single-hop routes requires linear time, the additional time is required to route the packets over multiple hops. Therefore the time required for the computation depends both on the number of connections sourced by a population and on the distance (or equivalently on the number of hops) between the source population and the destination population(s).

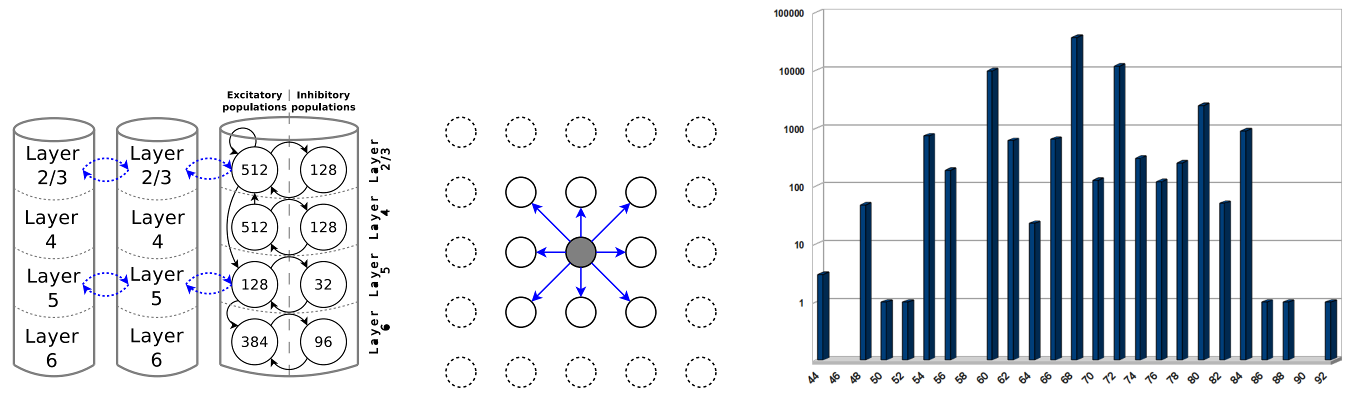
C. *The thalamocortical model*

In this last test we used a simplified biologically-inspired thalamocortical model [24], as described in fig.9(a). The figure shows the five-layer cortical columns we consider in this study. Circles represent populations, and the number of neurons in the population is denoted within. Black arrows represent the short-range connections, the ones within a single cortical column. Dotted arrows represent long-range connections between different cortical columns. A depiction of the structure of the long-range projections can be seen in fig.9(b) in which the circles represent cortical columns, and the arrows represent the projections among them. All the cortical columns follow the same structure of projections.

The system contains 512×512 total columns placed on a two-dimensional grid for a total of $2,097,152$ populations of neurons ($503,316,480$ neurons in total) and $7,327,752$ projections (long and short range). The size of the biggest population has been selected according to the number of neurons that each core is able to simulate, given the interconnection pattern designed. However, since this model has not been simulated, this size is a speculation. However, two factors makes this choice unimportant from the point of view of this test: first, the routing algorithm is independent of the exact size of the population, since the projection is represented by a single routing entry per population per projection. Secondly cortical columns have been chosen because they offer a good demonstration for scalable systems, and provide a good model to exercise the capabilities of the routing algorithm. We planned the model so that four cores are required to simulate a single column:

1) Excitatory population of layers 2/3 - Total 512 neurons;
2) Excitatory population of layer 4 - Total 512 neurons;
3) Excitatory populations of layer 5 (128 neurons) and layer 6 (384 neurons) - Total 512 neurons;
4) Inhibitory populations of layers 2/3 (128 neurons), layer 4 (128 neurons), layer 5 (32 neurons) and layer 6 (96 neurons) - Total 384 neurons;

While short range connections are kept local to each chip (each chip is able to accommodate 4 columns which are



(a) A collection of cortical columns. The rightmost shows the populations within each column.

(b) Depiction of the inter-cortical-column projections.

(c) Histogram of the number of entries in the routing tables.

Fig. 9. Details of the thalamocortical model test.

allocated sequentially from the pool of columns to place), long range connections do not take full advantage of the regularity of the SpiNNaker architecture because the 512×512 columns grid does not match the 256×256 structure of the SpiNNaker system. However some sort of repeating pattern will be present for long range connections.

The routing algorithm generated the routing tables for the $65,536$ routers in the system using between 92 and 44 entries for each router. The histrogram of the number of entries per router is described in fig.9(c): the horizontal axis represents the number of entries in the routers. The vertical axis is the number of occurrences of the specific number of routing entries in the whole SpiNNaker system. The tables for this experiment have been generated in slightly more than 7 hours using the population-based routing principles.

V. Discussion and future works

The three experiments described here provided the basis for an evaluation of population-based routing, with the particular goal of demonstrating that the time required to route all the projections, and the number of entries required in each routing table, allow the problem to be solved in a reasonable time. In particular the thalamocortical model experiment shows the result of routing using the principle that populations close together in the model reside in cores that are close to each other. Using a similar approach also for long-range connections may further reduce the number of entries in each router. In addition, in the case that the number of entries required is greater than the size of the routing table, it is possible to compress the entries further using a minimization tool.

Previous results about routing neuron-to-neuron have always been run for very small test systems (2×2 chips). An example of this has been published in [18] and took about two hours to generate the model for a networks of $4,000$ neurons and $225,000$ synapses. The problem with this approach is in the time required in the computation for bigger systems. In the neuron-to-neuron approach, the routing algorithm must be performed for each single projection. Therefore doubling the number of neurons results in a quadratic increase of the possible number of connections.

With this new approach the problem is moved up by one level of abstraction: the number of projections does not increase with the number of neurons in each population, but with the number of populations in the model. As a minor drawback, this algorithm requires an addressing space which is larger compared with the standard neuron-to-neuron communication channel definition (in the worst case the addressing space required is double the space strictly needed for the number of neurons simulated). However the SpiNNaker system has been designed to simulate a 1 billion neuron neural network, which is only $\approx 25\%$ of the addressing space offered by a 32 bit system, giving some additional space for a rational organization of the routing keys.

The algorithm described here is the first step towards a greater goal: a self-reconfiguring neuromorphic architecture. This algorithm has been designed to allow an implementation directly on the SpiNNaker system. In this way, during the setup phase, the SpiNNaker system will self-configure after a high-level description of the neural network. The routing algorithm described is only the first step of the complete process. Further steps include the generation of each synaptic connection required by the high-level description.

However, this is a first step in the direction of synaptic rewiring (also known as structural plasticity): during the neural simulation, when required from the biological model, it will be possible to connect two populations that initially were not supposed to be connected. This is a biological process that happens naturally at the beginning of life and with the acquisition of experience, but that has not yet been completely understood.

The SpiNNaker system has been designed to provide a tool for biologists to perform experiments and test hypothesis that emerge from the study of biological neural networks. The tool presented here is a step towards the implementation of such models in the SpiNNaker system with the objective of providing a tool to simulate such biological processes abstracting the description of the neural network up by one level to the population scale.

VI. Conclusions

This paper has described the multicast packet router in the SpiNNaker system and how the routing tables are computed off-line. This process, in fact, is now hosted on a PC that computes the routing tables starting from a high-level description of the neural network. In this context the "longest path first" algorithm has been chosen because it requires only local information to route packets to the destination chip (this algorithm requires only the source chip coordinates and the destination chip coordinates). The routing path may be computed hop by hop by each of the routers through which the packet has to pass, until the destination router is reached.

The outcome of this routing algorithm, however, shows that for both random networks and biologically-inspired networks, the number of entries used in the routing tables is rather low, as compared to the number of entries available. Therefore, even if the models used are designed only for testing purposes, the routing entries that are still available for each router enable the users to design networks that are more biologically detailed.

The new approach moves the problem of the distribution of the spikes in the system up of one level of abstraction: the number of projections does not depend any more on the size of each single population, but on the number of the populations simulated in the system. The computational improvements in this approach have allowed the routing of projections across a very large neural network, such as those for which the SpiNNaker system has been designed, in a reasonable time.

Acknowledgment

The SpiNNaker project is supported by the Engineering and Physical Science Research Council (EPSRC), grant EP/4015740/1, and also by ARM and Silistix. Dr. Navaridas holds a Newton International Fellowship from the Royal Society. We appreciate the support of these sponsors and industrial partners.

References

[1] S. Lefort, C. Tomm, J.-C. Floyd Sarria, and C. C. H. Petersen, "The Excitatory Neuronal Network of the C2 Barrel Column in Mouse Primary Somatosensory Cortex," *Neuron*, vol. 61, no. 2, pp. 301–316, Jan. 2009.

[2] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, V. J. Wedeen, and O. Sporns, "Mapping the Structural Core of Human Cerebral Cortex," *PLoS Biol*, vol. 6, no. 7, 2008. [Online]. Available: http://dx.doi.org/10.1371/journal.pbio.0060159

[3] T. Binzegger, R. J. Douglas, and K. A. Martin, "Topology and dynamics of the canonical circuit of cat V1," *Neural Networks*, vol. 22, no. 8, pp. 1071–1078, Oct. 2009.

[4] C. S. Lindsey and T. Lindblad, "Survey of neural network hardware," in *Proc. SPIE, Applications and Science of Artificial Neural Networks*, vol. 2492, 1995, pp. 1194–1205.

[5] L. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1-3, pp. 13–29, Dec. 2007.

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

[7] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, S. E. Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: a review of tools and strategies." *Journal of computational neuroscience*, vol. 23, no. 3, pp. 349–398, Dec. 2007.

[8] S. Draghici, "Neural networks in analog hardware–design and implementation issues." *International journal of neural systems*, vol. 10, no. 1, pp. 19–42, Feb. 2000.

[9] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, Dec. 2010.

[10] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, 1949.

[11] A. A. Fingelkurts, A. A. Fingelkurts, and S. Kähkönen, "Functional connectivity in the brain–is it an elusive concept?" *Neuroscience and biobehavioral reviews*, vol. 28, no. 8, pp. 827–836, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.neubiorev.2004.10.009

[12] M. James and D. Hoang, "Design of Low-Cost, Real-Time Simulation Systems for Large Neural Networks," *J. Parallel and Distributed Computing*, vol. 14, no. 3, pp. 221–235, Mar. 1992.

[13] S. B. Furber, S. Temple, and A. D. Brown, "High-Performance Computing for Systems of Spiking Neurons," in *Proc. AISB'06 workshop on GC5: Architecture of Brain and Mind*, Apr. 2006.

[14] A. D. Rast, X. Jin, F. Galluppi, L. A. Plana, C. Patterson, and S. Furber, "Scalable event-driven native parallel processing: the SpiNNaker neuromimetic system," in *Proceedings of the 7th ACM international conference on Computing frontiers*, ser. CF '10. New York, NY, USA: ACM, 2010, pp. 21–30.

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

[16] A. Rast, J. Navaridas, X. Jin, F. Galluppi, L. Plana, J. Miguel-Alonso, C. Patterson, M. Luján, and S. Furber, "Managing Burstiness and Scalability in Event-Driven Models on the SpiNNaker Neuromimetic System," *International Journal of Parallel Programming*, pp. 1–30, Jul. 2011.

[17] L. A. Plana, S. B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS infrastructure for a massively parallel multiprocessor," *Design & Test of Computers, IEEE*, vol. 24, no. 5, pp. 454–463, Oct. 2007.

[18] A. Rast, F. Galluppi, S. Davies, L. Plana, C. Patterson, T. Sharp, D. Lester, and S. Furber, "Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware," *Neural Networks*, vol. 24, pp. 961–978, 2011.

[19] J. Navaridas, M. Luján, J. M. Alonso, L. A. Plana, and S. Furber, "Understanding the interconnection network of SpiNNaker," in *Proceedings of the 23rd international conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 286–295.

[20] X. Jin, M. Lujan, L. A. Plana, S. Davies, S. Temple, and S. Furber, "Modelling of spiking neural networks on SpiNNaker," *Computing in Science and Engineering*, September/October 2010.

[21] 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," *Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2010.

[22] 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," *Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2010.

[23] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[24] A. M. Thomson and C. Lamy, "Functional maps of neocortical local circuitry." *Frontiers in neuroscience*, vol. 1, no. 1, pp. 19–42, Nov. 2007. [Online]. Available: http://dx.doi.org/10.3389/neuro.01.1.1.002.2007