# SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor

M.M. Khan, D.R. Lester, L.A. Plana, A. Rast, X. Jin, E. Painkras and S.B. Furber

***Abstract*— SpiNNaker is a novel chip – based on the ARM processor – which is designed to support large scale spiking neural networks simulations. In this paper we describe some of the features that permit SpiNNaker chips to be connected together to form scalable massively-parallel systems. Our eventual goal is to be able to simulate neural networks consisting of $10^9$ neurons running in 'real time', by which we mean that a similarly sized collection of biological neurons would run at the same speed.**

**In this paper we describe the methods by which neural networks are mapped onto the system, and how features designed into the chip are to be exploited in practice. We will also describe the modelling and verification activities by which we hope to ensure that, when the chip is delivered, it will work as anticipated.**

## I. Introduction

THE SpiNNaker Massively Parallel Computing System is designed to mimic neural computation [1] which is characterized by massive processing parallelism and a high degree of interconnection among the processing units [2]. The system is highly scalable to support a neural simulation from thousands to millions of neurons with varying degree of connectivity. A full scale computing system is expected to simulate over a billion neurons in real-time employing over one million processing cores to simulate neural computation in parallel. To support the tremendous amount of inter-neuron communication, a highly efficient asynchronous packet-switching routing network has been used to connect the processing cores. Figure 1 shows a logical view of the SpiNNaker computing system.

Inspired by the structure of brain, the processing cores have been arranged in independently functional and identical Chip-Multiprocessors (CMP) to achieve robust distributed computing. Each processing core is self-sufficient in the storage required to hold the code and the neurons' states, while the chip holds sufficient memory to contain synaptic information for the neurons in the system connected to the local neurons. The system at the chip- and system-level has sufficient redundancy in processing resources and communication infrastructure to provide a high level of fault tolerance. The platform makes a large-scale neural simulation possible in real-time which would otherwise take days to simulate with software on a personal computer.

As part of this research, a system-level functional model for the SpiNNaker computing system has been developed in SystemC capturing cycle-approximate functionality of all chip components at the transaction-level. Mapping neural applications on the SpiNNaker architecture as described in [3] has been tested on the system-level model with virtual inter-neuron connectivity by configuring the on-chip router to verify the functional correctness of the system while the components are still in the design phase. There were two mapping challenges as how to group the neurons to minimize inter-processor communication and then reduce the inter-chip communication; and how to keep the number of entries in the multicast routing table to a minimum. In this paper we share our experiences during the two case studies performed, mapping two different kinds of neural network applications onto the SpiNNaker architecture. The experiments will be of interest to the neural system engineering community, using architectures similar to the SpiNNaker [4].

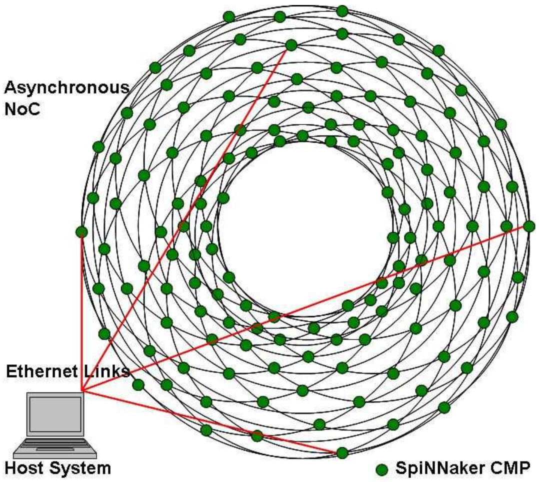

Fig. 1. Multichip SpiNNaker CMP System.

## II. SpiNNaker Architecture

The SpiNNaker CMP has been designed to support highly parallel distributed computing with high bandwidth inter-process communication. Each chip contains 20 ARM968 processing cores with a dedicated tightly-coupled memory holding 32KB of instructions and 64KB of data, sufficient to implement a fascicle (group of neurons with associated inputs and outputs) of about 1000 simple spiking neurons. Each processing core is provided with other peripherals such as a Timer, Interrupt Controller, Communication Controller and a DMA Controller to support neural computation. Besides the cores' private memory, each chip contains an off-chip

The authors are with the School of Computer Science, The University of Manchester, Manchester, UK (email: {khanm}@cs.man.ac.uk).

978-1-4244-1821-3/08/$25.00©2008 IEEE

SDRAM of 1GB which holds most of the information on synaptic connectivity. The SDRAM is shared among the 20 processing cores and transfers information to the tightly coupled memory of the core as and when required, using the DMA Controller. The off-chip memory module supports easy upgrading of memory to larger size as per the requirements of the application. The SDRAM is connected to the processing cores through the DMA Controller with the help of an asynchronous Network-on-Chip (NoC) as a high-bandwidth shared medium among the 20 processing cores [5]. The network is called the System NoC and it provides a bandwidth of 1Gb/s. Other chip resources, like the System RAM, Boot ROM, System Controller and the Router Configuration Registers are also connected to the processing cores *via* the System NoC.

To support the event-driven modeling of spiking neurons as described in [3], the Timer generates an interrupt with a millisecond interval notifying the processing core to update the neurons' state in its fascicle [6]. The other event is generated by the Communication Controller on receipt of a spike in the form of a multicast packet coming from a neuron in some other fascicle. The Communication Controller is also responsible for forming a (40bits) packet with source identifier (containing the firing neuron's identifier along with its fascicle identifier and chip address) as a routing key along with a control header byte on behalf of the firing neuron in the fascicle. The spike transmission is carried over yet another asynchronous NoC called the Communications NoC. The hub of this NoC is a specially designed on-chip multicast router that routes the packets (spikes) to 20 internal outputs corresponding to on-chip processing cores and 6 outward links towards other chips as a result of source-based associative routing. Each chip has links from its six neighbours that terminate in the Communications NoC where these along with the internal packets, are serialized by the Communications NoC before going to the router. The six two-way links on each chip extend the on-chip Communications NoC to the six other neighbouring chips to form a system-wide global packet-switching-network. A system of any desired scale can be formed by linking chips to each other with the help of these links, continuing this process until the system wraps itself around to form a toroidal mesh of interconnected chips as shown in Figure 1.

The router contains 1K words of associative memory as a look-up table to find a multicast route associated with the incoming packet's routing key. If a match is not found, the router passes the packet towards the link diagonally opposite to the incoming link. The process is called 'default routing' and it helps in reducing the number of entries in the look-up table. The global packet-switching network to support spike communication has been hierarchically organized to keep the address space to a manageable scale. On the global network, only chip addresses are visible. Each chip maintains a chip-level private subnet of 20 fascicle processing cores that is visible only to the local router, while an individual neuron's identifier is local to the processing core. With this scheme we can keep the global address space to a limited number of entries in the routing tables. The router masks the bits containing neurons identifier to look up against the chip address and fascicle identifier only. Fascicle identifier is included in this lookup to identify packets destined for the local chip. This way, for a fascicle size of 256 neurons we can mask 8 bits carrying the neuron's identifier and thus can reduce the number of entries in the routing table to only 64 instead of 16,384 (= $2^{14}$) with a source identifier of 14 bits as shown in Figure 2. To deal with transient congestion at the outer links, the router can route the packet to its adjacent link as a measure of emergency routing. The neighbouring chip's router aligns the packet back to its correct path. The router is an efficient hardware component that can route one packet per cycle at 200 MHz. The Communications NoC supports up to 6 Gb/s per chip bandwidth [7]. The router can also route point-to-point and nearest neighbour packets which are used for system management and debugging/configuration purposes.

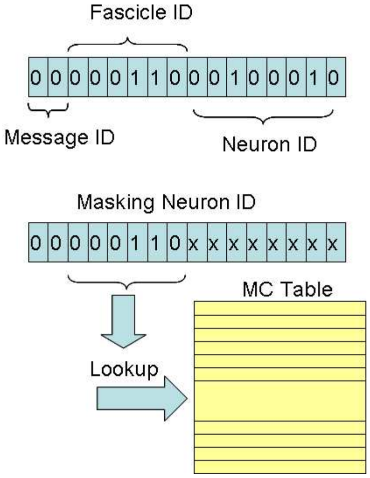

Fig. 2. Multicast Routing.

The system is connected to a personal computer called the Host System by linking any one (or more) chip(s) through an on-chip Ethernet link. The system is configured at the boot-up and the routing tables are computed and loaded to each chip as per the application's setting. The application is injected to the connected chip(s) using Ethernet frames, from where it is flood filled to other chips *via* the global asynchronous packet-switching network.

## III. Neural Network Models

Both biological data and computer modelling indicate that patterns of connectivity are largely responsible for observed differences in network behaviour. It is therefore essential to be able to model differing connectivity patterns in spiking neural hardware if the device is to be biologically plausible (or computationally useful), however, this presents significant challenges. Most significant among them is the sheer scale of the connection structure: even a relatively simple biological neural network might contain 10,000 neurons, with 1000 connections each, resulting in 10,000,000 potential connection paths for each neuron. Furthermore, known biological effects like Spike-Timing Dependent Plasticity (STDP) [8] and axonal delays [9] play significant roles in the computation that the model cannot ignore, or simplify too much, without risking severe degradation in realism or computing power. Implementing these connections as physical wires very quickly runs up against fundamental scaling barriers in silicon process technology: routing overhead, wire density, and limits to routability imposed by a finite number of metal layers. It is therefore essential to find novel routing strategies that permit a reduction in the number of wires, using virtual-mapping techniques that decouple the physical wire structure from the modelled connectivity of the network.

The solutions that present themselves in silicon technology fall into 2 broad classes, time multiplexing and component abstraction. Tuffy et al. [10] use time-multiplexing circuitry to aggregate the outputs from multiple neurons onto a single bus wire. This effectively collapses the interconnect, however, the neurons themselves remain physically implemented, and this in turn limits the achievable on-chip densities (to about 3000 neurons/chip), as well as hard-wiring some topological characteristics because of the fixed master/slave relationship of the bus architecture. Reconfigurable FPGA architectures, e.g. [11] are another, extremely popular time-multiplexing technique, here multiplexing the entire network by swapping out physical components. FPGAs are also popular for the second method, component abstraction [12]. The component abstraction technique reduces size by developing modules with generic functionality, which can implement arbitrary neural function (often with some simplification). FPGAs, however, are notorious for high power consumption, slow speed, and cumbersome reconfiguration. Hence application-specific hardware embedding dedicated abstract neural components have recently emerged [13]. These devices can be very general-purpose but still suffer from routing overhead unless combined with time-multiplexing techniques. However, if taken as completely arbitrary in topology and dynamics, such systems encounter formidable routability and memory size barriers.

Observed properties of real neural networks permit several important simplifying assumptions that make the general-purpose architecture feasible. One of the most significant is that neurons tend to cluster in well-identified groups possessing dense local connectivity and relatively sparse long-range connectivity [14]. Sometimes, e.g. in V1 regions in the visual cortex [15], the local connectivity patterns express via their spatial position external spatial or logical topologies in the real world; in other cases, e.g. hippocampal place cells [16], it is possible to identify connectivity groupings even though, to date, no direct mapping has been identifiable with external features. It is also evident from these observations that it must be possible to implement a hierarchical locality containing regions, subregions, fields, etc. Since in most cases, the projections of these regions into further processing areas are much sparser than the local connections, taking a model that presumes a connection density gradient with distance, particularly one with an exponential distribution, allows the mapping of neurons to processors and connections to concentrate associated groups into a single processor or a small group of processors on the same die, preserving limited long-range routing resources for the sparser connections.

Real spiking networks have both sparse patterns of activity and relatively slow signalling. Common estimates of spike frequency suggest an upper limit of about 100 Hz on spike-frequency rates, with delays over the entire network being of the order of milliseconds. By contrast silicon has delays of the order of nanoseconds, and if it is understood that the mapping technique decouples the physical wire structure from the connectivity patterns of the modelled network, the system will be able to hide delays and effects of the physical routing itself, so that the network model does not have any dependencies on the hardware implementation. Slow signalling speeds with sparse activity also mean the routers themselves can effectively process a large number of connections without excessive congestion. If about 1% of neurons are spiking at any one moment, at the full spiking rate of 100Hz, then a router processing at the rate of 100MHz will, on average, be able to handle the traffic from $10^8$ neurons. If this router contains 1000 routing entries, this means in turn that it will be able to aggregate $10^5$ neural signals into a single entry, greatly simplifying the structure since the router can simply steer generic groups of signals into a given path. In short, it is not particularly important that the router direct each individual signal along a precise route.

By taking advantage of these important characteristics, the hardware routing structure can, at least in theory, be relatively small and simple yet be able to cope with millions or even billions of connections without causing intractable difficulties in routability. The architecture, in essence, time-multiplexes large numbers of neural connections onto the same network fabric, so that a single physical link represents in fact multiple neural links, taking advantage of the great differences in speed between electronic and biological signal transmission. Biological experiments, however, demonstrate that delays in neural networks, particularly axonal delays, have a significant effect on the processing, and this would be a potentially fatal shortcoming of the architecture if it were proposed that the network fabric itself model these delays, either through direct hardware support or by a scaling of the silicon delays themselves. In a separate paper [6], however, we demonstrate that it is possible to model these delays effectively at the processing nodes themselves, taking advantage in this case of the extreme difference in the processing speed of digital computation against real neurons. Therefore, no part of the network fabric itself need have any direct correlation to the network being modelled, and this is the essence of the approach, *complete decoupling of the silicon hardware from the modelled network.*

Such a network is flexible enough to map multiple different patterns of connectivity in the same way that various different biological networks or brain regions display markedly different connectivity patterns, e.g., the difference between cerebellar regions and primary visual cortex. At the same time it does not exhibit any critical model dependence: the hardware does not embed the characteristics of a particular neural model into its design, and so it is able to accomodate new and developing models of neuronal computation, e.g. [17]. Provided the routing structure is run-time reconfigurable, it can also dynamically reassign connections, not simply by a reshuffling of weight values (although this is a simple way to achieve changes) but also through processes that physically remap connections operating over long time scales, which recent work demonstrates may be important for long-term development [18]. Ideally, such connectivity should be able to express the inherently fault-tolerant property of biological systems: the network can adapt and remap itself to compensate for failure or death of a group of neurons.

Critically, however, the connectivity must be scalable. Biological neural networks range from simple systems of creatures such as the squid up to the extraordinary complexity of the human brain, and yet use the same underlying architecture, the same componentry, the same signalling methods. If a system for spiking neural networks is to be biologically plausible, therefore, it must display at least reasonable scalability, so that the same mapping and routing strategies that are effective for small groups of tens or hundreds of neurons remain effective when modelling millions or tens of millions of neurons. It is first and foremost the question of scalability that makes it essential to turn to simulation and modelling, in order to verify the architecture prior to committing it to silicon.

## IV. Mapping Neural Networks

To test the hardware models and to check that our routing table software (SpiNNit) works correctly, we have developed a mechanism to generate random networks and to generate random traffic on such networks. One quick and obvious way to generate test data is to ignore the virtualization of the addresses of the neurons and instead work with neurons that have already been placed in a fixed 2-dimensional square grid pattern. It is then a simple matter to connect neurons randomly together. This is the approach we have taken, since we then do not have to contemplate the problem of assigning locations to virtual neurons. Subsequently, we also generate random packets for this network and with these we validate the hardware design.

We begin by describing our method for generating random neural nets. The worst case for the SpiNNaker is when neurons connect to other neurons with no locality effects. This can be modelled by connecting a neuron to any other neuron with fixed probability $p$. The distribution of the connections is then *uniform* since the probability of connecting to any other neuron is constant. Simulation data for this model was encouraging in the sense that the machine still ran, but it needs to be borne in mind that the simulated systems were relatively small. Although the *performance* of the machine was adequate the routing tables were small. This behaviour arose because most spikes were sent to most of the other processors, and hence default routing and multicast provide an effective method for getting spiking events to all of the other processors that would need to know what had happened.

However, the SpiNNaker machine is designed to perform best when neural connections demonstrate some kind of locality. One simple way to generate the required locality is to generate connections for which there is a higher probability of a short connection than a long one. A simple mathematical model of this assumes that the distribution of connections is normally distributed in the $x$ and $y$ directions. This can be thought of as a 'target-like' distribution where most of the 'hits' are clustered around the centre, and fewer 'hits' occur in the outlying regions. The distribution of connections is then *normal*, since the dependence of connection probability with the $x$ and $y$ distances follows a normal Gaussian distribution.

The method for generating these normally distributed random connections uses the Box-Muller transformation [19]. If we consider the pair of random variables $(X, Y)$ as describing rectangular coordinates in 2-dimensional space then the transforming them to polar coordinates makes things much easier.

Suppose that $U_1, U_2$ are independent, continuous, uniformly-distributed random variables on the interval $(0, 1]$. Then, letting $r = \sqrt{-2\ln(U_1)}$ and $\theta = 2\pi U_2$, we obtain two independent, normally-distributed random variables $X$ and $Y$ with $X = r\cos(\theta)$ and $Y = r\sin(\theta)$; each is distributed $N(0, 1)$.

This also demonstrates how to compute a sequence of exponentially-distributed random numbers: generate a uniformly-distributed random variable ($U$) and then transform to $X = \ln(U)$. This is exponentially-distributed with parameter $\lambda = 1$.

An interesting phenomenon has been observed: networks with *high* locality generate routing tables with relatively few entries (6–12); networks with *low* locality generate routing tables with relatively few entries (again 6–12); however, networks with intermediate locality generate very large routing tables (typically $1,000$–$10,000$ entries). We can handle this behaviour by multicast default routing these spikes to all neurons, but this is less than ideal. It remains to be seen if this behaviour is a feature of real neural nets or is instead an artifact of simplistic modelling assumptions.

To generate random traffic, we have given each neuron a fixed time constant, and assumed that it fires after a time $t$ which is exponentially distributed. This simplifies the mathematical modelling – queueing theory with exponential inter-arrival times is tractable – but the assumption that these events are independent is obviously unlikely to be observed in reality.

Our mathematical modelling of the generation of networks and traffic is crude, and it remains for the hardware to be built in order to see how accurate we have been. It should be noted that the methods described have been useful debugging and validation tools, even if the data generated is not entirely accurate.

## V. Implementation Process for Mapping the Neural Networks

As shown above, each SpiNNaker chip is connected to six neighbours. We expect a *flat*, 2D interconnect – resulting in a hexagonal pattern – to suffice for the intended application and this will allow straightforward layout on PCBs. However, this certainly should not be taken to imply that the system can only model 2D neural structures; SpiNNaker can model networks in two, three or more dimensions. The key to this flexibility is that SpiNNaker maps each neuron into a virtual address space. Assignments can be arbitrary, though assignments related to physical structure are likely to improve modelling efficiency. Neurons can be allocated to any processor, and the routing tables must be configured to send the neural events accordingly.

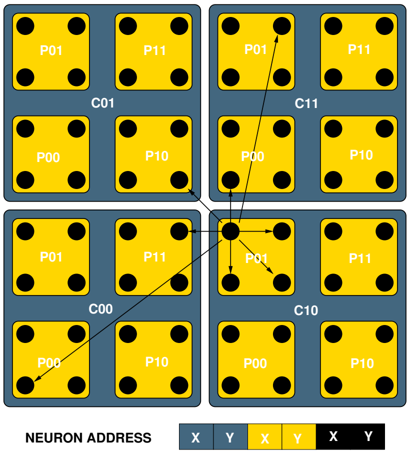

Fig. 3. Neuron Assignment and Mapping.

The first step in the mapping process is to assign every neuron to a processor. Given the absolute flexibility in assignment, this is a heuristically-driven step. Figure 3 shows a small (8 neuron by 8 neuron, *shown as black circles*) section of a 2D neural network. Spiking neural nets tend to show locality of connections, *i.e.*, neurons tend to be connected to nearby neurons, forming clusters; these clusters are sometimes known as fascicles. Therefore, assigning neighbouring neurons to the same processor, or processors in the same chip, will result in shorter routes and less inter-chip traffic. Figure 3 shows a very simple, 'rectangular' assignment of neurons to processors (*labelled Pnn*) and chips (*labelled Cnn*). Interestingly, even though the chips are connected in a pattern that would suggest some form of hexagonal addressing scheme [20], our experiments have shown that a Cartesian coordinate system is adequate for an efficient mapping and routing process.

The second step in the process is to map the neuron into a virtual address space. As explained above, the routers can associate the neurons in groups and each group is routed using the same look-up table entry. The chosen mapping, shown at the bottom of Figure 3, guarantees that neurons that have been assigned to the same processor can be grouped in the same routing entry. By giving proper values to the router entry masks, neurons assigned to different processors in the same chip can also be grouped in the same entry, if adequate.

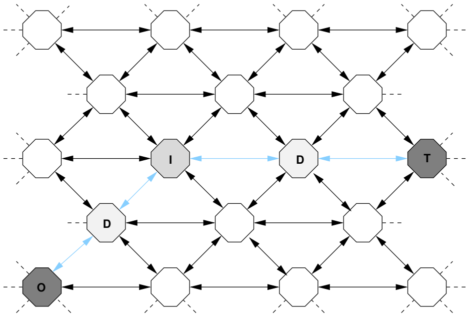

Fig. 4. Route Setup.

After the neurons have been assigned to processors and mapped into the virtual address space, routing information can be generated. The first step is to set up a route for every connection in the neural net. The process is driven by two criteria: the route should go through the minimum number of routers, and should result in the minimum number of router entries. These criteria should optimize the use of router resources and should also have a positive impact in network traffic. The 'default' routing mechanism, introduced above, is used to eliminate the need for routing entries in most routers along the selected route. An example which shows how routes are constructed (or 'setup') is illustrated in Figure 4. A packet sent from the 'origin' node (labelled O in the figure) to the 'target' node (labelled T) traverses the predefined route shown. The route is one of the (possibly many) shortest routes available, which will comprise, at most, two straight segments that meet at the 'inflection' or turning node (labelled I). The segments may contain intermediate nodes (labelled D). If a multicast packet is sent along the route described above, routing entries are needed in nodes O, I and T, while the rest of the nodes, *i.e.* D nodes, can take advantage of the default routing mechanism. Default routing can reduce the size of the routing tables significantly, especially in long-distance neuron connections.

Once a route has been set up for every connection associated with a neuron, the next step combines the individual routing entries into multicast routing entries. This must be done carefully, given that some of the individual routes rely on default routing. As a result of this step, 'primitive route tables' are generated, so called because each table has, at most, one routing entry per neuron. In most cases, due to the locality of connections and the use of default routing, routers will not require a routing entry for a large majority of the neurons.

The final step in the process generates minimal routing tables. For this purpose, the tables are treated as logic functions: the multicast entries constitute the on-set of the function, the default entries constitute the off-set and 'unrelated' neurons, *i.e.*, those that have no routes traversing the node, are considered the 'don't-care' set. Currently espresso [21], a well-established logic minimizer, is used to minimize the tables. An application called SpiNNit was developed to automate the process and this was used in the generation of the results presented later.

## VI. Implementation

The system-level model for the SpiNNaker computing system captures the architectural details for each component in the SpiNNaker CMP to exhibit a cycle-approximate behaviour at the functional level. For asynchronous components like the Communications NoC and the System NoC, the timing behaviour has been incorporated into the model using Hardware Description Language (HDL) behavioural simulation. The ARM core has not been simulated with an Instruction Set Simulator (ISS); however, it maintains cycle-approximate communication at the transaction-level with all its peripherals using the AMBA High-performance Bus (AHB) model. The model is flexible in scale, memory size, address space and clock timing, all of which can be configured with parameters. We simulated the system-level model for the SpiNNaker computing system at various scales to verify functional hypotheses. The neural mapping as described later was implemented on the system-level model for two different types of neural problems helping us to verify its functionality at multiple levels. The mapping was implemented in C++ to run on the processing cores' models while the routing tables entries for each on-chip router were computed with the help of SpiNNit, developed as a part of this exercise.

### A. Simple Spiking Neurons Model

This was a very simple case study designed to verify the router's functionality. For this we mapped one neuron to each fascicle processor. Each chip contained four fascicle processors and one monitor processor while the system consisted of 4x4 (16) chips. The neurons' mapping information was provided to the SpiNNit to determine the routing table entries for the on-chip routers' multicast lookup and mask tables. Sample packet files for each processor were created by the SpiNNit with expected output files containing the packets each fascicle processor should receive at the end of the simulation. The simulation recorded the output packets with one output file for each processor. While the fascicle output files recorded the received packets, monitor processor output files contained any packet dropped to the monitor processor due to some error such as long lasting congestion, parity error and time-phase error. It was difficult to check all the output files manually against their expected output files and then trace any unexpected or dropped packet to the monitor processor. An automated process was created with the help of the transaction recording functionality of the SystemC Verification (SCV) library. This compared the output with the expected outputs and generated a report for any missing or unexpected packets found in any fascicle output file. The experiment was run several times with many variations for debugging and verification purposes. The results matched the expected behaviour. The tests included emergency routing, default routing and packet dropping to the monitor processor. Critical to reducing routing table size is the successful exploitation of default routing. This remains true in the presence of congestion with emergency routing, as the emergency routing mechanism brings the packet to its original path despite having been diverted.

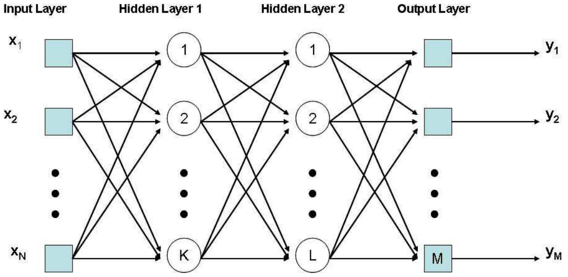

Fig. 5. A Typical Multilayer NN Model.

### B. Layered Neural Networks Model

A typical multilayer neural network learning algorithm consists of an input layer, an output layer and a number of hidden layers each containing neurons with the connectivity as shown in Figure 5. Psychologists use a similar model: the Parallel Distributed Processing (PDP) learning model. We carried out a case study to see the feasibility of using SpiNNaker for running the PDP learning model with the researchers in the School of Psychological Sciences, at the University of Manchester. They have been using a PC Cluster for running the simulation with the Light Efficient Network Simulator (LENS) [22] application. Every neuron in a layer can receive input from many neurons (depending on the connectivity level) in its preceding layer. The inputs are multiplied by the connection weights and then summed together to produce the output for the next layer. The output layer neurons add the weighted inputs to generate the activation using the activation function. The output is used with the target output to compute the error (delta) to backpropagate synaptic updates. The PDP model is very different from a spiking neural model, as the packets have to carry payload as input to the next layer neurons and there is no notion of real-time involved. Moreover, many inputs can converge at the same target neuron at the same time, which might have caused bottleneck for the Communications NoC.

These variations made the task a bit difficult for the communication system of SpiNNaker; however, we could use

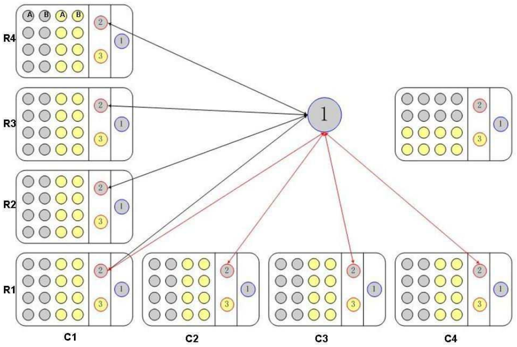

Fig. 6. PDP Neural Mapping.

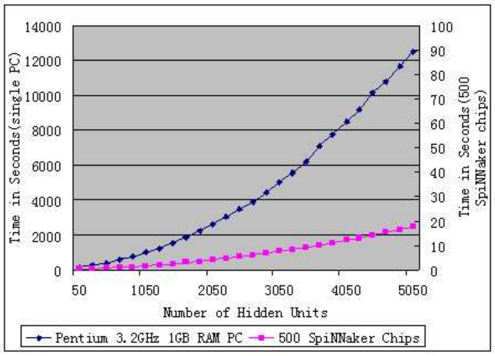

Fig. 7. Simulation of the PDP Model on the SpiNNaker System.

the flexibility in the SpiNNaker architecture to circumvent this problem. Instead of using a direct, one-unit-to-one-processor mapping with the inherent risk of traffic congestion at target units, we cast the problem as a weights matrix multiplication problem of the neurons in the hidden and the output layers. Consider the neurons as partial result-computing units and feed the results forward to the next layer units which accumulate these results to those further ahead, until received by the output layer to compute the activation and delta. The same process is repeated in back-propagation of the delta.

We divided the large matrix computational problem into small portions, assigning one portion to each processor out of the many spread over different chips. To further reduce the router traffic we decided to pass on results among the processors on the same chip by using a shared-memory message-passing technique with the help of on-chip System RAM. The partial results to the processors outside the chips were passed as multicast packets after configuring the routing tables as per the procedure described above. As shown in Figure 6, the dark coloured and light coloured processors in processor columns marked as A in each chip compute partial products from the input vector and the weight matrix, and pass these to their corresponding coloured processors marked as number 2 and 3 respectively using shared-memory message-passing technique. Processors number 2 and number 3 on each chip in the chip columns C1 to C4 send their results to processors number 1 in rows R1 and R2 respectively using multicast packets. After a specified period of time the same procedure is repeated for dark and light coloured processors in processor columns B on each chip. The processors number 1 are the output layer processors that compute the activation output and the partial delta values. In the second phase of feed-forward pass, processor 1 of each chip in chip-column C1 will multicast the partial delta result to all number 2 processors in row R1, processor 1 in column C2 will multicast to all number 2 processors in row R2 and so on. During back-propagation, the computed delta is transmitted backward following the same path.

The routing tables for forward and backward path multicast packets were computed with the help of SpiNNit for various scales of the SpiNNaker system. Because the ARM968 core has been modeled at an abstract level in the SpiNNaker system-level model, the computation was modeled using the cycle-accurate ARM RealView ARMulator ISS model with an ARM968 core to acquire accurate computational timing including the time for shared-memory message passing between local chip processors. We implemented the PDP model in C++ on the system-level model incorporating the processing delays acquired from RealView ARMulator. We simulated a 195-chip SpiNNaker computing system for 150,000ns of activity. As the model gives accurate timing for the communication, the resulting simulation time was approximately what we expect on the SpiNNaker hardware. The simulation results with a comparison to LENS simulation on a PC are shown in Figure 7. In this case study also, we managed to keep the routing table entries to a minimum with the help of default routing, masking and shared-memory message-passing techniques among the fascicle processors on the same chip.

## VII. Conclusions

We have investigated an alternative to direct-mapped hardware neural networks: soft-mapping neural connectivity using a hardware structure that permits a virtual network topology to be superimposed on a dedicated architecture designed for its mapping flexibility. Where with direct-mapped approaches the principal challenges lie in achieving the required wiring densities in silicon, with the routed virtual network approach the behaviour and resource requirements depend on the degree of inhomogeneity of the dynamics and the topology. Highly homogeneous networks, while easily mapped, tend to strain the processing capability of the system since their heavy, symmetric loads defeat attempts to localise processing or routing. Highly inhomogeneous networks are also easily mapped, but utilise resources inefficiently since the extremely local nature of the processing in this case makes it difficult to distribute the work. Intermediate cases, meanwhile, efficiently use resources but can strain the size of the mapping tables required, because the system cannot make use of simplistic default routing that aggregates large numbers of connections. From a system standpoint, this divides the mapping space into distinct regions or 'phases', where the behaviour is extremely different, and suggests that the nature of the mapper required will likewise use a multiphase algorithm to optimise the routing tables.

Initial simulations of performance, while encouraging, indicate that it will be essential to simulate larger networks. How the performance scales with very large systems containing thousands of processors is at this point unclear; what preliminary testing shows is that the effectiveness of the mapper in finding optimal neuron placement and routing will have an impact on system performance sufficiently great that it will probably form the difference between a working system and one that does not work. Presently we are refining our system-level model to enable application development using the ARM instruction set, incorporating a cycle-accurate instruction set simulator (ISS) model of the target ARM968E-S processing core with its other peripherals. With this model we can develop and test the intended neural applications for the target hardware. For this purpose the Izhikevich [23] spiking neuron model, already implemented in an ARM968E-S ISS using ARM's proprietary Realview ARMulator [6] is being ported to our system-level simulator. Provided the test chip is available once this model is complete, the developed applications will then be run on it for hardware verification. Performance analysis on the basis of a system-level simulation of a reasonable scale will be carried out in near future for design verification. The same application, after fine tuning, shall serve as a golden model to refine our mapping technique discussed in this paper. We intend also to study dynamic mapping by readjusting routing tables at run-time based on the synaptic update information at each chip and congestion on the inter-chip links.

We have shown that the system is able to model very different classes of neural network, but for this capability to be useful, it must operate in conjunction with configuration software that can likewise optimise the mapping for the particular class of network. There thus remains considerable work to be done in developing the configuration tools, identifying and classifying biologically realistic patterns of connectivity, and working both on software simulation and hardware modelling to extend the size of the networks under simulation. SpiNNaker's virtual network approach to neural modelling extends the size and class of neural networks that hardware can model, but as an effective tool requires new insights into the nature of neural connectivity, insights that may in time prove as useful to the biological community that is the target end user of the system as to the computational community that is the source developer of it.

## Acknowledgements

The Spinnaker project is supported by the Engineering and Physical Sciences Research Council, partly through the Advanced Processor Technologies Portfolio Partnership at the University of Manchester, and also by ARM and Silistix. Steve Furber holds a Royal Society-Wolfson Research Merit Award. We appreciate the support of these sponsors and industrial partners.

## References

[1] S. Furber, S. Temple, and A. Brown, "On-chip and inter-chip networks for modelling large-scale neural systems," in *Proc. International Symposium on Circuits and Systems, ISCAS-2006*, Kos, Greece, May 2006.

[2] P. Dayan and L. Abbott, *Theoretical Neuroscience*. Cambridge: MIT Press, 2001.

[3] S. B. Furber, S. Temple, and A. D. Brown, "High-performance computing for systems of spiking neurons," in *AISB'06 workshop on GC5: Architecture of Brain and Mind*, vol. 2, Bristol, April 2006, pp. 29–36.

[4] S. Furber and S. Temple, "Neural Systems Engineering," *J. R. Soc. Interface*, vol. 4, no. 13, pp. 193–206, Apr. 2007.

[5] A. Rast, S. Yang, M. Khan, and S. Furber, "Virtual synaptic interconnect using an asynchronous network-on-chip," in *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[6] X. Jin, S. Furber, and J. Woods, "Efficient modelling of spiking neural networks on a scalable chip multiprocessor," in *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[7] 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," *IEEE Design & Test of Computers*, vol. 24, no. 5, pp. 454–463, Sept.-Oct. 2007.

[8] H. Markram and M. Tsodyks, "Redistribution of Synaptic Efficacy Between Neocortical Pyramidal Neurons," *Nature*, no. 382, pp. 807–810, 1996.

[9] S. Crook, G. Ermentrout, M. Vanier, and J. Bower, "The role of axonal delay in the synchronization of networks of coupled cortical oscillators," *J. Computational Neuroscience*, vol. 4, no. 2, Apr. 1997.

[10] F. Tuffy, L. McDaid, M. McGinnity, J. Santos, P. Kelly, V. W. Kwan, and J. Alderman, "A time-multiplexing architecture for inter-neuron communications," in *Proc. 2006 Int'l Conf. Artificial Neural Networks (ICANN 2006)*, 2006, pp. 944–952.

[11] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization," *IEEE Trans. Neural Networks*, vol. 18, no. 3, pp. 880–888, May 2007.

[12] M. Porrmann, U. Witkowski, H. Kalte, and U. Rückert, "Implementation of artificial neural networks on a reconfigurable hardware accelerator," in *Proc. 2002 Euromicro Conf. Parallel, Distributed, and Network-based processing*, 2002, pp. 243–250.

[13] R. Eickhoff, T. Kaulmann, and U. Rückert, "SIRENS: A simple reconfigurable neural hardware structure for artificial neural network implementations," in *Proc. 2006 Int'l Joint Conf. Neural Networks (IJCNN2006)*, 2006, pp. 2830–2837.

[14] O. Shefi, I. Golding, R. Segev, E. Ben-Jacob, and A. Ayali, "Morphological characterization of in vitro neuronal networks," *Phys. Review E*, vol. 66, no. 2, Aug. 2002.

[15] K. Ohki, S. Chung, Y. H. Ch'ng, P. Kara, and R. C. Reid, "Functional imaging with cellular resolution reveals precise micro-architecture in visual cortex," *Nature*, no. 433, Feb. 2005.

[16] D. Zipser, "A computational model of hippocampal place fields," *Behavioral Neuroscience*, no. 99, 1985.

[17] E. Izhikevich, "Polychronization: Computation with spikes," *Neural Computation*, vol. 18, no. 2, Feb. 2006.

[18] T. Elliott and N. Shadbolt, "Developmental robotics: Manifesto and application," *Philosophical Trans. Royal Soc.*, vol. A, no. 361, 2003.

[19] M. Kendall, A. Stuart, and O. J. K., *Advanced Theory of Statistics*, 5th ed. London: Charles Griffen and Company Ltd., 1987, vol. 1.

[20] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing, and broadcasting in hexagonal mesh multiprocessors," *IEEE Transactions on Computers*, vol. 39, no. 1, pp. 10–18, Jan. 1990.

[21] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. on Computer-Aided Design*, vol. 6, no. 5, pp. 727–750, Sep. 1987.

[22] D. L. T. Rohde, "Lens: The light, efficient network simulator," *From. http://www.cs.cmu.edu/ dr/Len - Retrieved 17 June 2007*, 1999.

[23] E. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. on Neural Networks*, vol. 14, pp. 1569–1572, Nov. 2003.