Biologically-Inspired Massively-Parallel Architectures – computing beyond a million processors

Steve Furber
The University of Manchester,
Oxford Road, Manchester M13 9PL UK
steve.furber@manchester.ac.uk

Andrew Brown
The University of Southampton,
Southampton, Hampshire SO17 1BJ UK
adb@ecs.soton.ac.uk

Abstract

The SpiNNaker project aims to develop parallel computer systems with more than a million embedded processors. The goal of the project is to support large-scale simulations of systems of spiking neurons in biological real time, an application that is highly parallel but also places very high loads on the communication infrastructure due to the very high connectivity of biological neurons. The scale of the machine requires fault-tolerance and power-efficiency to influence the design throughout, and the development has resulted in innovation at every level of design, including a self-timed inter-chip communication system that is resistant to glitch-induced deadlock and 'emergency' hardware packet re-routing around failed inter-chip links, through to run-time support for functional migration and real-time fault mitigation.

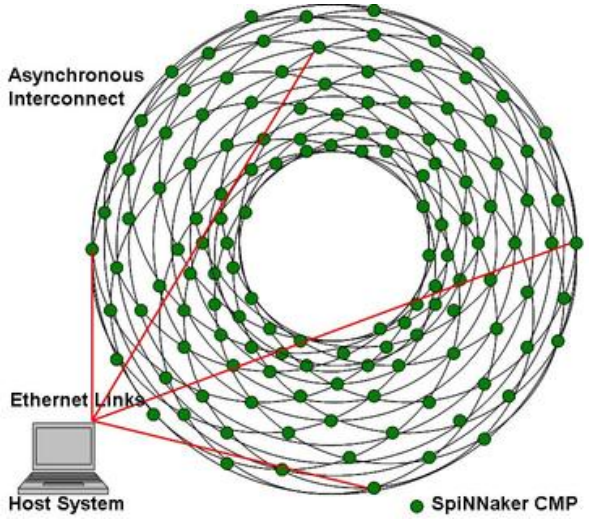

Figure 1. The SpiNNaker system

1. Introduction

The human brain remains as one of the great frontiers of science – how does this organ upon which we all depend so critically actually do its job? A great deal is known about the underlying technology – the neuron – and we can observe large-scale brain activity through techniques such as magnetic resonance imaging, but this knowledge barely starts to tell us how the brain works. Something is happening at the intermediate levels of processing that we have yet to begin to understand, but the essence of the brain's information processing function probably lies in these intermediate levels. To get at these middle layers requires that we build models of very large systems of spiking neurons, with structures inspired by the increasingly detailed findings of neuroscience, in order to investigate the emergent behaviours, adaptability and fault-tolerance of those systems [1].

The goal of the SpiNNaker project is to deliver machines of unprecedented cost-effectiveness for this task [2], and to make them readily accessible to as wide a user base as possible. We will also explore the applicability of the unique architecture that has emerged from the pursuit of this goal to other important neural models [3] and other application domains.

The approach taken towards this goal is to develop a massively-parallel computer architecture based on Multi-Processor System-on-Chip (MPSoC) technology (see Fig. 1) capable of modeling a billion spiking neurons in biological real time, with biologically-realistic levels of connectivity between the neurons.

The machine will incorporate more than a million ARM processor cores, but despite this level of computing power and the simplified neuron models the architecture is optimized for, the billion neuron objective represents only 1% of a human brain. It is hoped, however, that simulations on this scale will deliver new insights into the fundamental mysteries of brain function. Such insights might allow the level of abstraction used in the models to be raised, but at this stage this is little more than speculation.

In order to maximize the cost-effectiveness of the machine we must control both the build and the running costs.

2. The computer architecture perspective

Multi-core processors are now established as the way forward on the desktop, and highly-parallel systems have been the norm for high-performance computing for some time. In a surprisingly short space of time, industry has abandoned the exploitation of Moore's Law through ever more complex uniprocessors, and is embracing the 'new' Moore's Law: the number of *processor cores* on a chip will double roughly every 18 months. If projected over the next 25 years this leads inevitably to the landmark of a million-core processor system.

Much work is required to understand how to optimise the scheduling of workloads on such machines, but the nature of this task is changing: in the past, a large application was distributed 'evenly' over a few processors and much effort went into scheduling to keep all of the processor resources busy; today, the nature of the cost function is different: processing is effectively a free resource. Although the automatic parallelisation of general-purpose codes remains a 'holy grail' of computer science, biological systems achieve much higher levels of parallelism, and we turn for inspiration to connectivity patterns and computational models based on our (limited) understanding of the brain.

This biological inspiration draws us to two parallel, synergistic directions of enquiry; significant progress in *either* direction will represent a major scientific breakthrough:

- How can massively parallel computing resources accelerate our understanding of brain function?
- How can our growing understanding of brain function point the way to more efficient parallel, fault-tolerant computation?

We start from the following question: what will happen when processors become so cheap that there is, in effect, an unlimited supply of them? The goal is now to get the job done as quickly and/or energy-efficiently as possible, and as many processors can be brought into play as is useful; this may well result in a significant number of processors doing identical calculations, or indeed nothing at all - *they are a free resource*.

What sets the SpiNNaker work apart from other many-core systems is the neurological inspiration behind the architecture and the very fine granularity of our processing. We use smaller processors in greater numbers than other machines, with significant benefits in terms of cost and, in particular, energy-efficiency. Two metrics determine the cost-effectiveness of a many-core architecture:

- MIPS/mm^2 – how much processing power can a unit of silicon area yield?
- MIPS/W – how much energy does it take to execute a given program?

On the first of these measures embedded and high-end processors are roughly equal – a SpiNNaker chip with 20 ARM cores delivers about the same throughput as a high-end desktop processor – but on energy-efficiency the embedded processors win *by an order of magnitude*.

The drawback of fine granularity is generally in mapping the application to the system, but here we start knowing that one of our key applications – modelling large systems of spiking neurons – maps readily, because the architecture is moulded to this application. What we find, but have yet to prove, is that the principles that make the neural model efficient can be applied to other problems. Those principles are *bounded asynchrony* (time models itself), *virtualised topology* (physical and logical connectivity are decoupled), and *energy frugality* (processors are free; the real cost of computing is energy). We will expand on these principles in Section 3.

2.2 Living with failure

The relative importance of fault-tolerance as a design parameter is set to grow considerably. Well before feature sizes approach quantum limits, point reliability will decrease markedly, not least because of process spread and the consequent variability of component characteristics. Designers are increasingly tasked with building reliable systems out of fundamentally unreliable components. Current reliability engineering (of non-global scale systems) involves considerable resource duplication (be it memory or processing or interconnect) and rarely, if ever, removes every single-point-of-failure. The internet, originally conceived as a computer cluster

with no single vulnerable point, has (probably) achieved this, but it is truly global in scale, and even so, analyses have shown that – planet wide – it is probably vulnerable to a coincidence of less than a dozen specific failures.

Biological neural systems, on the other hand, achieve massive fault tolerance at the 'device' level. The average adult human loses a neuron *every second of their lives* and there is usually no discernable behavioural consequence. In contrast, how many interconnects in a PC must be cut before it stops working?

The neural architecture embodied in SpiNNaker will not solve this conundrum at a stroke, but it will provide a platform for research in this area.

3. Design principles

3.1 Bounded asynchrony

One of the three key ideas behind the SpiNNaker architecture is that *time models itself*: time is free running and there is no global synchronization. This is sometimes referred to as "bounded asynchrony" since interactions are asynchronous but threads progress at very similar rates. Physical systems have this characteristic: nature has no global clock but the rate of *local* change is controlled by *local* physical effects. This timing model contrasts with most traditional supercomputing models where communication between threads is highly synchronous and deterministic.

Within the SpiNNaker machine this is implemented through the use of real-time event-driven application code. Neuron models are computed in real time, which typically requires accuracy on millisecond timescales. A millisecond timer event in each processor causes the neuronal differential equations to be evaluated, possibly resulting in a spike output event. Spike events generate small packets that are delivered well within a 1ms time window to any target processor in the system, but system-wide (approximate) synchrony is just a side-effect of the 1ms timer interrupts running at the same rate throughout the system and the communication delays being negligible on the ms timescale.

3.2 Virtualized topology

The second key idea behind SpiNNaker is that *physical and logical connectivity are decoupled* [4].

Because communication is effectively instantaneous (on a biological timescale) throughout the system there is no requirement for the mapping of

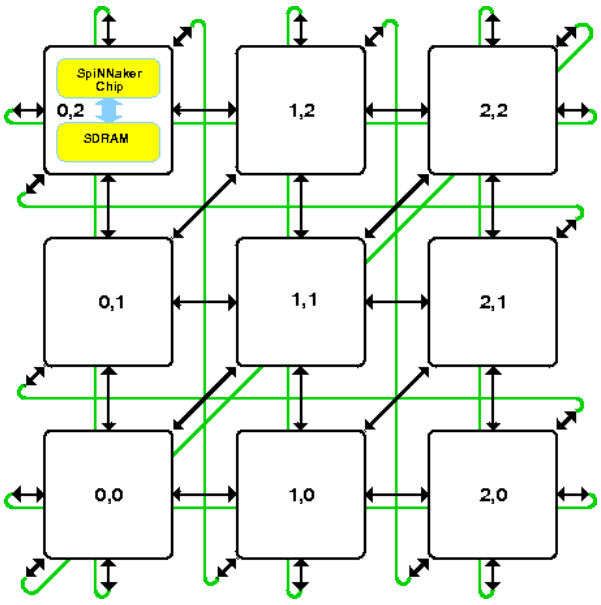

Figure 2. SpiNNaker mesh detail

neurons onto SpiNNaker to follow their physical topology in any way. In principle any neuron can be mapped onto any processor. In practice it is likely to be beneficial to map neurons that are physically close in biology to proximal locations in SpiNNaker as this will minimize routing costs, but it is not necessary to do so. This allows SpiNNaker to exploit a simple two-dimensional topology to model three-dimensional biological structures. Abstract higher-dimensional neural structures could also be accommodated provided they operated with communication delays similar to three-dimensional biological systems, where the delay from one neuron to the next increases with the Euclidian distance between them.

The downside of the (biologically-) instantaneous electronic communication is that the delays in biological systems are almost certainly functional, so they can't simply be eliminated in the model. Instead, they are made 'soft'. Each synapse has a programmable delay associated with its input, which is re-inserted algorithmically at the target neuron [5]. This is, in fact, one of the most expensive functions of the neuron models in terms of the cost of data storage held locally to the processor that models the target neuron.

3.3 Energy frugality

The third key idea behind SpiNNaker is that *processors are free; the real cost of computing is*

energy. This has not been true historically, but the cost of processors is falling and the cost of energy is rising. If this trend continues (as seems likely), then this assertion will become true at some point.

Where are we now? A PC costs around $1,000 and consumes 300W. A Watt costs $1/year. So the energy cost of a PC equals the purchase cost after a little more than three years, which is of the same order as the typical useful life of a PC. At current prices the purchase and energy costs are roughly equal, so the above assertion is on the verge of becoming true.

Embedded processors can reduce the capital and energy costs of a given level of compute power by about an order of magnitude, thereby significantly reducing the ownership (and environmental) costs. The embedded processor technology employed in SpiNNaker delivers a similar performance to a PC from each 20-processor node, for a component cost of around $20 and a power consumption under 1 Watt.

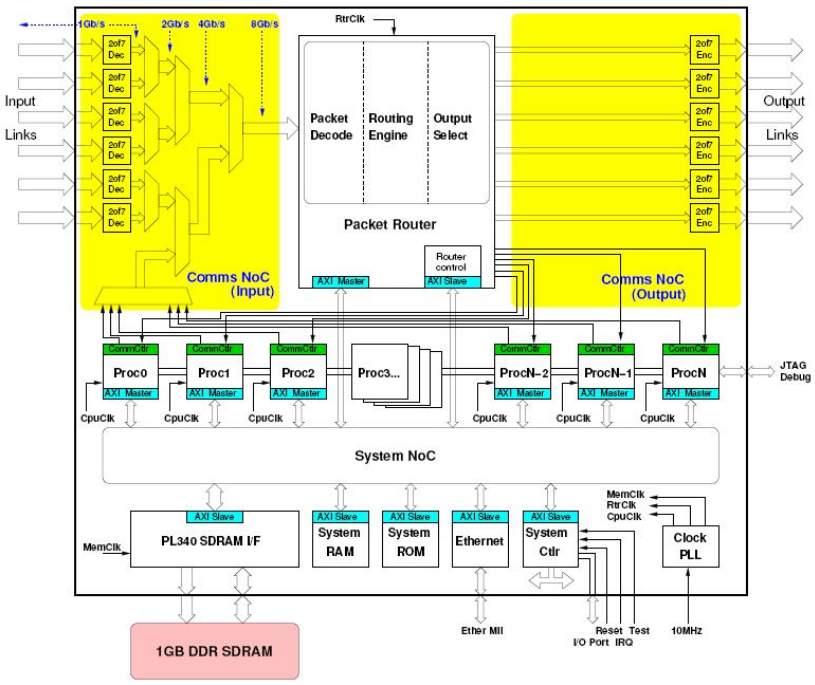

Figure 3. A SpiNNaker node

4. SpiNNaker system architecture

SpiNNaker is conceived as a two-dimensional toroidal mesh of chip multiprocessors (CMPs) connected via Ethernet links to one or more host machines (Fig. 1). The 2-D mesh has triangular facets (Fig. 2) to support 'emergency routing' around a failed or congested link. Each CMP node comprises two chips: a SpiNNaker MPSoC and a 1Gbit mobile DDR SDRAM memory. The SpiNNaker MPSoC incorporates up to 20 ARM968 processors, each with local memory and support peripherals, interconnected by two self-timed Network-on-Chip (NoC) fabrics developed using tools and libraries supplied by Silistix Ltd [6] (Fig. 3).

- The *Communications NoC* carries neural spike event packets between processors on the same and different nodes.
- The *System NoC* is used as a general-purpose on-chip interconnect to allow processors to access system resources such as the shared SDRAM.

The feature of the architecture that renders it uniquely suited to modeling large-scale systems of spiking neurons is the Communications NoC and the associated multicast Packet Router [7] on each node. Each neuron that emits a spike causes its host processor to generate a 40-bit packet that contains 8 bits of packet management data and a 32-bit identifier of the neuron that fired. Identifying neuron spikes by using a unique identifier for the source neuron is known as *Address Event Representation* (AER), and has been used for some time by the neuromorphic community [8][9]. In the past AER has been used principally in bus-based broadcast communication between neurons, but here we employ a packet-switched multicast mechanism to reduce total communication loading.

Each processor subsystem (Fig. 4) has 32 Kbytes of instruction memory, 64 Kbytes of data memory, a timer/counter, a vectored interrupt controller, a communications controller (which sends and receives neural event and other types of packet) and a DMA controller that is typically used to transfer blocks of synaptic connectivity data from the SDRAM to the processor local memory in response to the arrival of an incoming neural spike event.

Although the processor subsystem itself employs a conventional clocked synthesized RTL design flow, all of its interfaces are self-timed. This means that timing closure issues are contained within this relatively small

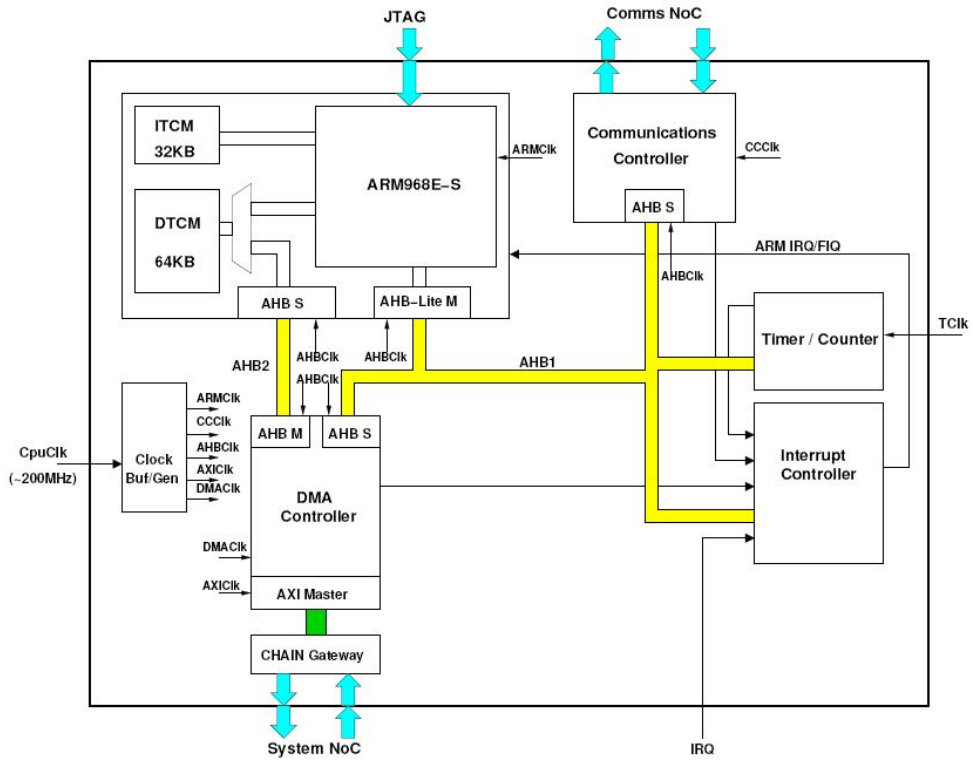

Figure 4. A SpiNNaker processor subsystem

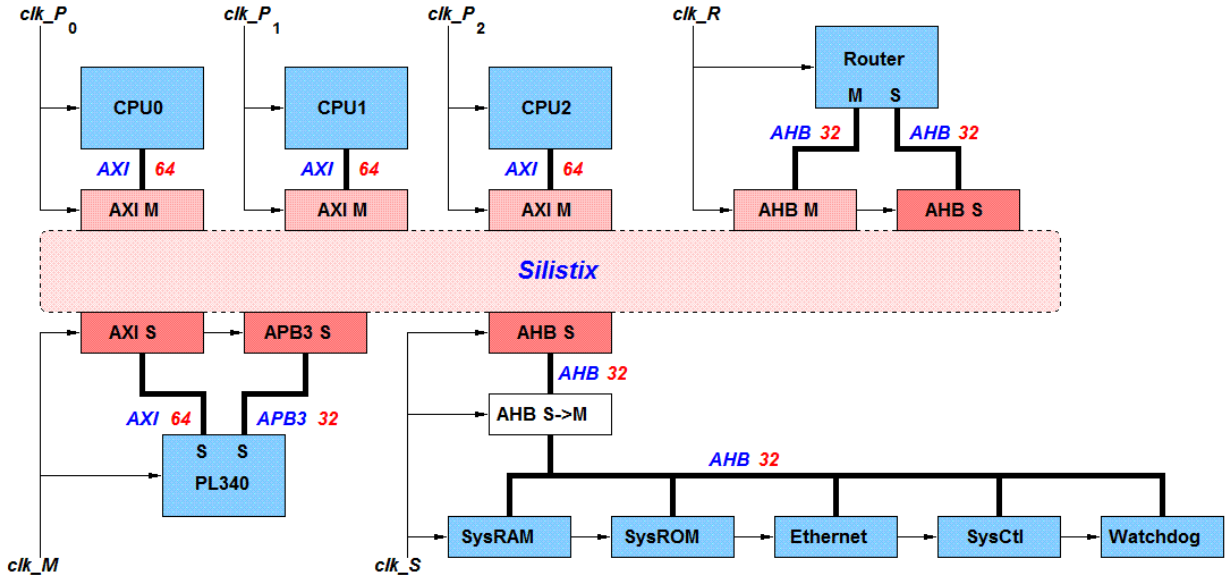

Figure 5. SpiNNaker GALS organization

design component and do not spread upwards to full chip level. The complete chip (and, indeed, system) can be seen to be an example of *Globally Asynchronous Locally Synchronous* (GALS) design practice [10][11] as illustrated in Fig. 5. A benefit of the GALS approach, in addition to the simplification of

timing closure during the design process, is that it decouples the clocks and power supply voltages at each of the clocked submodules, offering flexibility to the designers in coping with, and optimizing for, the increasing process variability expected in future deep submicron manufacturing processes. The GALS approach is also capable of supporting traffic service management [12].

5. Concurrency

SpiNNaker is a massively-parallel system that embodies concurrency at every level in the design. At each level there are new challenges to address. Some of these are described in the following subsections.

5.1 Circuit-level concurrency

The on- and inter-chip communications fabrics employ a range of self-timed Network-on-Chip structures and protocols [13]. Within the individual clock domains, the interconnect is based on ARM's AMBA protocols, using a combination of AXI, AHB and APB bus standards as determined by a combination of system requirements and what is available in the 'off-the-shelf' IP blocks provide by ARM. The global on-chip interconnect is developed using tools and libraries from Silistix Ltd which employ 3-of-6 return-to-zero (RTZ) self-timed codes and an interconnect switching fabric based on CHAIN [6]. The inter-chip links employ 2-of-7 non-return-to-zero (NRZ) self-timed codes that effectively bridge the CHAIN NoC across from one chip to another.

The choice of a different protocol for the chip-to-chip link was motivated by two considerations:

- *performance*: an RTZ protocol requires two complete chip-to-chip out-and-return signaling paths to be completed for each symbol, one for the outgoing symbol and acknowledgement, and a second for the return to zero and its acknowledgement. An NRZ protocol only goes round this loop once per symbol, effectively doubling the throughput.
- *power*: a 2-of-7 NRZ code uses 3 off-chip wire transitions to send 4 bits of data; a 3-of-6 RTZ code uses 8 wire transitions to send the same 4 bits.

So in the off-chip domain, where chip-to-chip delays dominate performance and wire transitions dominate power consumption, the 2-of-7 NRZ code delivers twice the performance for less than half the energy per 4-bit symbol of the 3-of-8 RTZ code used on chip. In the on-chip domain the balance is very different, and the simpler logic of the RTZ code dominates the decision on both power and performance.

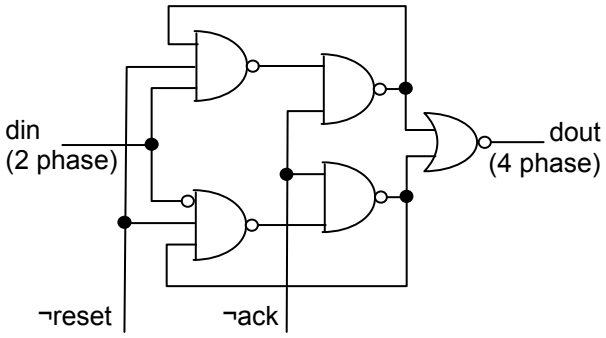

Figure 6. Phase converter

A difficulty with self-timed protocols is that they are very 'brittle' in the presence of transient or permanent faults. As a first step in addressing this weakness, the SpiNNaker inter-chip links have been designed to tolerate transient faults (in the form of injected glitches) on the inter-chip wires. It is not possible to avoid data corruption, so the goal is to minimize the risk of deadlock resulting from glitch injection, and to simplify recovery from deadlock as much as possible.

A 2-phase communication protocol will maintain phase parity in the absence of faults, and conventional circuit implementations rely on this to recover data by XORing the level of a wire carrying 2-phase data with locally-generated state to recover the 4-phase data value. However, such an implementation is prone to lose state in the presence of faults, resulting in deadlock. We therefore use a true transition-sensing data recovery circuit (Fig. 6) that is insensitive to phase parity errors [14]. This circuit also ignores further transitions on its data input until it is re-enabled by the acknowledge signal (¬ack), thereby protecting downstream circuits from additional spurious inputs. This circuit, together with a number of other circuit enhancements, has reduced the occurrence of deadlocks in our glitch simulations by a factor 1,000, indicating that the circuit will keep passing data (albeit with errors) in the presence of quite high levels of interference on the inter-chip wires.

At a higher level in the circuit we wish to be able to reset subcircuits so that, should deadlock occur, we can recover with minimal disruption. Here there is a fundamental issue: the inter-chip link can be viewed as a cycle with a single token that is passed from end to end. If we reset one end how do we avoid destroying

the token, thereby causing deadlock, or creating a second token that leads to circuit malfunction?

The solution we have adopted is to cause both transmitter and receiver circuits to inject a token when they exit from reset, thereby deliberately creating the 2-token problem when both ends are reset at the same time, and we rely on the ability of the circuit in Fig. 6 to absorb (and ignore) a second token that arrives while it is awaiting data to send with the first token.

5.2 System-level concurrency

SpiNNaker is a highly-distributed homogeneous system with no explicit means of synchronization. This creates a number of problems at boot and application-load times. These problems are addressed using a number of techniques that ensure that the machine can be bootstrapped and an application loaded in reasonable time [15].

Firstly, each chip contains 20 processors in a symmetric configuration, but one of these is set aside as *Monitor Processor* to perform system management functions. The choice of Monitor Processor is not fixed in the hardware for reasons of fault tolerance; instead all processors perform self-test at start-up and then all those that pass the test can bid to serve as Monitor. There is a read-sensitive register in the System Controller that effectively serves as arbiter in this process, ensuring that one and only one processor is chosen as Monitor. This breaks the symmetry in the on-chip structure, and thereafter the Monitor Processor is in control and carries out the remaining boot and application load functions.

The interconnect fabric and Router support three different packet types. One has already been discussed – the *multicast* (mc) packet type that conveys neural spike event information. The other two types are:

- *Point-to-point* (p2p) packets, which are used to convey system management information. These have conventional 16-bit source and destination addresses and are routed algorithmically.
- *Nearest-neighbour* (nn) packets, which allow processors on one chip to communicate with any of the six chips to which there is a direct connection.

If any node fails to boot correctly its neighbours will detect this and can attempt to remedy the problem. Using nn packets they can change the choice of Monitor Processor, and they can copy boot code into the failed node's System RAM and instruct it to reboot from there.

The second phase of the boot process involves breaking symmetry at the system level. One of the nodes connected by Ethernet to the Host System (see Fig. 1) is identified as the origin, and is given the coordinates (0,0). This positional information is then propagated throughout the system using nn packets to allow each node to determine its position in the 2-D mesh (Fig. 2). Only then can each node configure its p2p routing tables, which allows the Host System to communicate with any node using p2p packets via Ethernet and node (0,0).

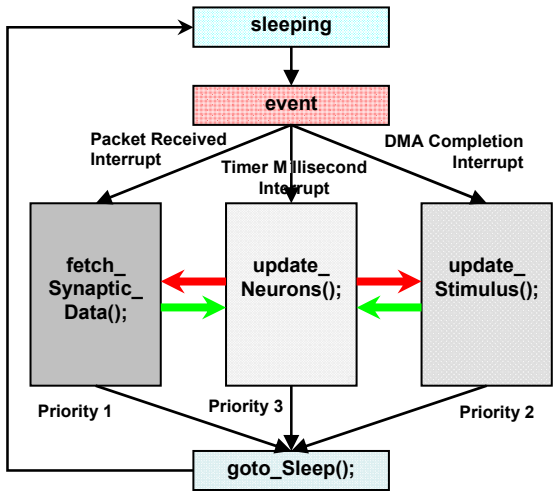

Figure 7. Event-driven real-time model

Now the system is ready for an application, which is loaded using flood-fill techniques and nn packets. The flood-fill mechanism has been shown to give load times almost independent of the size of the machine, with trade-offs between load time and the degree of fault-tolerance, which can be controlled by the number of times a node receives each component of the application [15].

5.3 Application-level concurrency

The SpiNNaker application model is real-time and event-driven. Every active application processor (which excludes the Monitor Processors, and any processor which is idle or disabled due a suspected fault) is executing the same three tasks (possibly with different local algorithms [16]) in response to interrupt events (Fig. 7). When all tasks are completed the processor goes into a low-power 'wait for interrupt' state. The three tasks are:

- *Incoming packet arrival*. When an application is running, application processors will receive only mc packets that convey real-time information about a neural spike event. The processor must identify the spiking neuron, map this to the associated block of connectivity data in SDRAM, and then schedule

a DMA to load that information in the processor's local data memory.

- *DMA complete*. A block of connectivity data has been loaded, so the processor can initiate the next scheduled DMA transfer and process the connectivity data. This processing may generate output neural spike events and, if the connectivity data is modified, a DMA must be scheduled to write the changes back into SDRAM.
- *1ms timer interrupt*. This is used to compute the real-time dynamics of the differential equations that describe the neuron dynamics, and also to schedule inputs delayed due to local modeling of axonal delays [17].

Mapping the biological neural system onto the SpiNNaker machine is non-trivial [18][19]. Neurons must be mapped to processors, multicast routing tables computed, connectivity data constructed, and relevant input/output mechanisms deployed. We have simple examples of all of these running on models of the system, but scaling them up to the full million processor machine will require a great deal of work and probably the development of new algorithms.

Once loaded and running the application then demands a real-time response across the system. The communications fabric is designed to deliver mc packets in significantly under 1ms, whatever the distance from source to destination. It is also intended to operate in a lightly-loaded regime to minimize congestion and the additional delays that that would incur. However, neural spike traffic is bursty, and we cannot rule out at design time that peak traffic might cause transient congestion. In addition, the failure of an inter-chip link will cause major local congestion.

In order to minimize the impact of transient congestion or link failure, the SpiNNaker routing engine includes a low-level hardware 'emergency routing' mechanism [7]. This allows the Router to sense when packets have stopped flowing through a link. Fig. 8 illustrates the normal route taken by a packet from its origin ('O') through a point of inflection ('I') to its target ('T'), passing through a node that employs simple pass-through default routing ('D') on each leg. Consider the case when link 'a' becomes congested. After a programmable delay the Router will invoke emergency routing to redirect packets that should pass through the affected link around the two other sides ('b' and 'c') of one of the mesh triangles of which the affected link is the third side. If the problem is transient the link will unblock in due time, and normal flow will resume. In any case the local Monitor Processor can be informed about the invocation of emergency routing and can decide whether some additional intervention is required to avoid congestion recurring, or to find a permanent rerouting around a failed link.

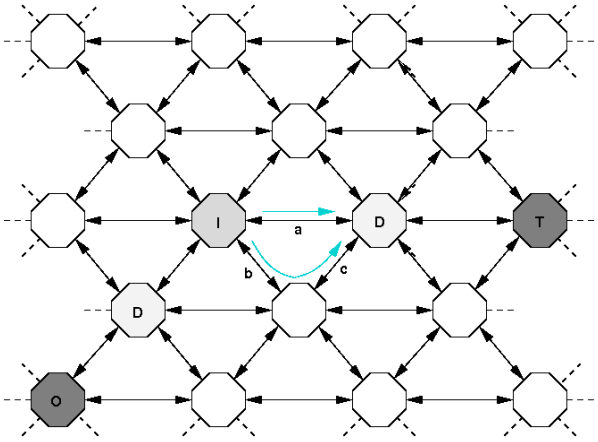

Figure 8. Emergency routing example

A standard issue in high-performance parallel computing is the avoidance of deadlock in the inter-processor communications fabric. This is often achieved by avoiding any circular loops in the communication fabric. The SpiNNaker communication fabric makes no attempt to avoid loops, but it must still guarantee to avoid deadlocks. The solution we have adopted is simple, and exploits the intrinsic fault-tolerance of the neural applications we aspire to support: no Router will get into a state where it persistently refuses to accept incoming packets, and if necessary it will simply drop outgoing packets in order to achieve this. Thus, if an output link is blocked, first the Router waits a programmable time, then it tries emergency routing for a programmable time, then it gives up and drops the packet. The local Monitor Processor is informed of the failure, and can recover the packet and re-issue it if appropriate.

5.4 Biological concurrency

Of greatest interest in this work is, of course, the fundamental question of how concurrency is exploited in the biology that we are trying to model. The brain is itself a massively-parallel system comprising low-performance asynchronous components. Those components, neurons, operate at timescales of a millisecond or greater, and the primary means of information exchange is through the emission of electrical 'spike' events. These spikes seem to carry no information in their amplitude or impulse, they are pure asynchronous events that carry information only in the time at which they occur.

So, how is information represented in the brain? The oldest theory is that information is encoded as the rate of spiking of a neuron. But this description alone is insufficient to explain the speed of response of, for example, a human subject to a new visual stimulus, where there is time for any neuron in the several neural layers through which the information must pass to fire no more than once. It is hard to estimate a firing rate from a single spike! Alternative theories have emerged that suggest that the information may be encoded in the choice of a subset of a population that is active at any time, which in its purest form is an N-of-M code familiar to the asynchronous design community (though with N and M values in the hundreds or thousands, rather than the low units as is common in engineered systems). In an extension of this approach, the N active neurons convey additional information in the order in which they fire – these are 'rank-order' codes [20]. There is then the question of how the start and end of a particular salvo of spikes is determined. One possible answer is in the observation of background rhythms in many parts of the brain; it is possible that each rank-order salvo occurs on the rising surge of a rhythm, and the falling phase of the rhythm acts as a symbol separator. But now we are well into the domain of speculation.

In the better-understood areas of the brain, such as the retina and early vision processing areas of the cortex, it seems that each spiking neuron has a receptive field within which sensory inputs or earlier processing layers affect the output rate either positively or negatively. In the retina, for example, the spiking ganglion cells have characteristic centre-on surround off ('Mexican hat') or centre-off surround-on receptive fields, representing an array of two-dimensional filters that are applied to the image on the retina [21]. The filters cover the retina at different overlapping scales, and lateral inhibition reduces the information redundancy in the resultant stream of spikes that passes through the million or so fibres that form the optic nerve.

If a neuron fails it will cease to generate output and also cease to generate lateral inhibition, so a near-neighbour with a similar receptive field will take over and very little information will be lost. This may go someway towards explaining the remarkable fault-tolerance of the brain, which continues to function normally despite the loss of around one neuron per second throughout adult life. Of course, the second trick up biology's sleeve is that the brain is not a fixed network; it can adapt to events such as neuron failure and re-optimize the surrounding circuits to compensate for the lost functionality.

6. Conclusions

The SpiNNaker project aims to deliver cost-effective parallel computing resources at an unprecedented scale, with over a million embedded processors delivering around 200 teraIPS to support the simulation of a billion spiking neurons in biological real time. The scale of the system demands that power-efficiency and fault-tolerance feature prominently among the design criteria, and the result is a design that embodies concurrency at all levels, from circuit through system to application.

Many challenges remain, since we have yet to gain access to silicon so many of the ideas presented here have yet to be proven "in the flesh". But we have exposed the design to extensive simulation, up to and including running real-time spiking neuron application code on SystemC [22] and Verilog RTL models incorporating four chips and eight ARM968 processors, so we are gaining considerable experience in driving the system in full detail. There is still a long way to go, however, before the machine will be ready for use by neuroscientists and psychologists who do not wish to have to contend with concurrency issues at any level below the neurological model that they wish to simulate.

Understanding how the brain develops, learns and adapts remains as one of the Grand Challenges of science. A new computer, however powerful and well-adapted to running models of the brain, is not going to solve this mystery in a single step. It is merely a tool alongside a whole range of other tools in neuroscience and psychology that offer perspectives on the problem. As an example of concurrency in action, however, the brain is supreme, and new theories of concurrency are likely to be required before any real understanding of the principles of operation of the brain can emerge, to complement the modeling capabilities of machines such as SpiNNaker and many other ongoing developments in neuroscience and psychology.

7. Acknowledgements

The SpiNNaker technology is being developed within research projects funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grants GR/S61270/01, EP/D07908X/1 and EP/G015740/1, in collaboration with the universities of Southampton, Cambridge and Sheffield, and with industrial support from ARM, Silistix and Thales. Steve Furber is supported by a Royal Society-Wolfson Research Merit Award. We are grateful for the support received from these sources.

Engineering is a team sport, and many members of staff and students have contributed to the development of SpiNNaker. We would like to acknowledge the contributions from Steve Temple, Luis Plana, Eustace Painkras, Viv Woods, Jim Garside, Dave Lester, David Clark, Peter Wilson, Mark Zwolinski, Jeff Reeve, Mikel Lujan, Alex Rast, Yebin Shi, Jian Wu, Mukaram Khan, Xin Jin, Shufan Yang, Cameron Patterson, Dom Richards, Zhenyu Liu, Francesco Gallupi, Javier Navaridas, and others!

8. References

[1] S.B. Furber and S. Temple, "Neural systems engineering", *J. Roy Soc Interface* **4**(13), 2007, pp. 193-206.

[2] S.B. Furber, S. Temple and A.D. Brown, "High-performance computing for systems of spiking neurons", *Proc. AISB'06 vol. 2*, Bristol, 2006, pp. 29-36.

[3] A.D. Rast, S.R. Welbourne, X. Jin and S.B. Furber, "Optimal connectivity in hardware-targetted MLP networks", *Proc. IJCNN*, Atlanta. Georgia, 14-19 June, 2009 (to appear).

[4] A.D. Rast, S. Yang, M.M. Khan and S.B. Furber, "Virtual synaptic interconnect using an asynchronous Network-on-Chip", *Proc. IJCNN*, June 2008, pp. 2727-2734.

[5] A.D. Rast, X. Jin, M.M. Khan and S.B. Furber, "The deferred event model for hardware-oriented spiking neural networks", *Proc. ICONIP*, Auckland, New Zealand, Nov 25-28, 2008.

[6] W.J. Bainbridge and S.B. Furber, "CHAIN: A delay insensitive CHip Area Interconnect", *IEEE Micro* **22**(5), 2002, pp. 16-23.

[7] J. Wu, S.B. Furber and J.D. Garside, "A programmable adaptive router for a GALS parallel system", *Proc. Async'09*, Chapel Hill, North Carolina, 17-20, May 2009 (to appear).

[8] M. Mahowald, *VLSI analogs of neuronal visual processing: a synthesis of form and function*, Ph.D. dissertation, California Inst. Tech., Pasadena, CA, 1992.

[9] M. Sivilotti, *Wiring considerations in analog VLSI systems, with application to field-programmable networks*, Ph.D. dissertation, California Inst. Tech., Pasadena, CA, 1991.

[10] 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 and Test of Computers* **24**(5), 2007, pp. 454-463.

[11] D. Chapiro, *Globally Asynchronous Locally Synchronous Systems*, Ph.D. Thesis, Stanford University, 1984.

[12] S. Yang, S.B. Furber, Y. Shi and L.A. Plana, "An admission control system for QoS provision on a best-effort GALS interconnect", *Proc. ACSD*, 2008, pp. 200-207.

[13] S.B. Furber, S. Temple and A.D. Brown, "On-chip and inter-chip networks for modelling large-scale neural systems", *Proc. ISCAS*, Kos, Greece, 21-24 May 2006, pp. 1945-1948.

[14] Y. Shi, S.B. Furber, J.D. Garside and L.A. Plana, "Fault-tolerant delay-insensitive inter-chip communication", *Proc. Async'09*, Chapel Hill, N. Carolina, 17-20 May 2009 (to appear).

[15] M.M. Khan, J. Navaridas, X. Jin, L.A. Plana, M. Lujan, J.V. Woods, J. Miguel-Alonso and S.B. Furber, "Event-driven configuration of a neural network CMP system over a homogeneous interconnect fabric", *Proc. ISPDC*, Lisbon, Portugal, June 30 - July 4, 2009 (to appear).

[16] A.D. Rast, M.M. Khan, X. Jin, L.A. Plana and S.B. Furber, "A universal abstract-time platform for real-time neural networks", *Proc. IJCNN*, Atlanta. Georgia, 14-19 June, 2009 (to appear).

[17] X. Jin, S.B. Furber and J.V. Woods, "Efficient modelling of spiking neural networks on a scalable chip multiprocessor", *Proc. IJCNN*, June 2008, pp. 2812-2819.

[18] A.D. Brown, D.R. Lester, L.A. Plana, S.B. Furber and P.R. Wilson, "SpiNNaker: the design automation problem", *Proc. ICONIP*, Auckland, New Zealand, Nov 25-28, 2008.

[19] M.M. Khan, D.R. Lester, L.A. Plana, A.D. Rast, X. Jin, E. Painkras and S.B. Furber, "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor", *Proc. IJCNN*, June 2008, pp. 2849-2856.

[20] R. Van Rullen and S. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex", *Neural Comput.* **13**(6), 2001, pp. 1255–1283.

[21] B.S. Bhattacharya and S.B. Furber, "Evaluating Rank-order Code Performance Using A Biologically Derived Retinal Model", *Proc. IJCNN*, Atlanta, Georgia, 14-19 June, 2009 (to appear).

[22] M.M. Khan, E. Painkras, X. Jin, L.A. Plana, J.V. Woods and S.B. Furber, "System-level modelling for SpiNNaker CMP system", *Proc. RAPIDO'09*, Paphos, Cyprus, January 25, 2009.