A communication infrastructure for a million processor machine

A.D.Brown[1], S.B. Furber[2], J.S. Reeve[1], P.R.Wilson[1], M. Zwolinski[1], J. E. Chad[3], L. Plana[2] and D. Lester[2]
1: Electronics and Computer Science, University of Southampton, SO17 1BJ UK
2: Computer Science, University of Manchester, M13 9PL UK
3: School of Biological Sciences, University of Southampton, SO17 1BJ UK

Abstract:
The SpiNNaker machine is a massively parallel computing system, consisting of 1,000,000 cores. From one perspective, it has a place in Flynns' taxonomy: it is a straightforward MIMD machine. However, there is no interconnecting bus structure, and there is no attempt to maintain coherency between any of the memory banks. Inter-core communication is implemented by means of chip-to-chip packet transfer. Unlike conventional parallel machines, where packet-based communication is supported by a software layer, in SpiNNaker, the packet communication fabric is built in at the hardware level, and is the only mechanism whereby an arbitrary pair of cores can communicate. There is no unifying synchronisation system, and the packet delivery infrastructure is non-deterministic.
In a number of application arenas - most notably neural simulation - this architecture is remarkably powerful, and supports techniques that can only be clumsily realised in conventional machines.
In order to realise these advantages, a software layer - the routing system - is necessary to facilitate and choreograph the movement of the packets throughout the machine. The sheer size of the SpiNNaker machine makes conventional techniques difficult or impossible; the machine has to be largely self-organising. The routing tables that underpin the communication infrastructure can therefore be derived dynamically as a prologue processing phase. This paper describes the low-level software packet management system embodied in SpiNNaker.

I. Introduction

SpiNNaker is a massively parallel computing machine, consisting principally of a million ARM 9 cores, configured as 20 cores/die on 50,000 chips. Detailed architectural descriptions and discussions may be found in [1-7], and - referees willing - elsewhere in this conference. Whilst the engineering of a million processors is no mean feat, in order that this (or any large multi-core) ensemble functions in a meaningful and efficient manner, it is necessary to provide some mechanism for these processors to communicate, both between themselves and to the outside world. The scale of the machine demands novel approaches to many aspects of these problems, and here we allow the machine to derive the routing tables (the core elements of the communications subsystem) for itself, without reference to any guiding input from outside. This paper describes the large-scale software communications infrastructure employed in the SpiNNaker machine.

I. 1 SpiNNaker - what is it for?

From one perspective, SpiNNaker is a parallel computing engine, containing a million cores, and is as programmable as any other parallel machine. However, it has been designed with one very specific goal in mind: the simulation of extremely large neural circuits, *in real time*. This goal has driven almost all of the major design decisions.

Each core has associated with it a small amount of program and data memory (the ARMs are configured in a Harvard architecture), and each chip also has a much larger tranche of memory to be shared amongst the twenty cores. Each chip communicates to the others via six high-speed bidirectional IO ports, which communicate with the processors via the routing infrastructure described in this paper. The chips are nominally phase-locked by a relatively low frequency clock, but *no attempt is made to enforce any further coherency on the memory or cores on each chip*. The rationale for this is simple: the overarching goal of the SpiNNaker system is the simulation of large aggregates of biological neurones in real time. The biological system being simulated is demonstrably (a) viable and stable, and (b) non-deterministic, so the value of the simulation is not affected if the machine itself introduces - within limits - some non-determinism. Removing the constraint of determinism makes the design (from the computer architect perspective) around an order of magnitude simpler. The impact on the efficiency of the simulation process has yet to be studied in detail. Each ARM core hosts the state of around 1000 neurons, making the SpiNNaker machine capable of simulating aggregates of around a billion neurones - in real time, assuming a real neurone has a peak firing

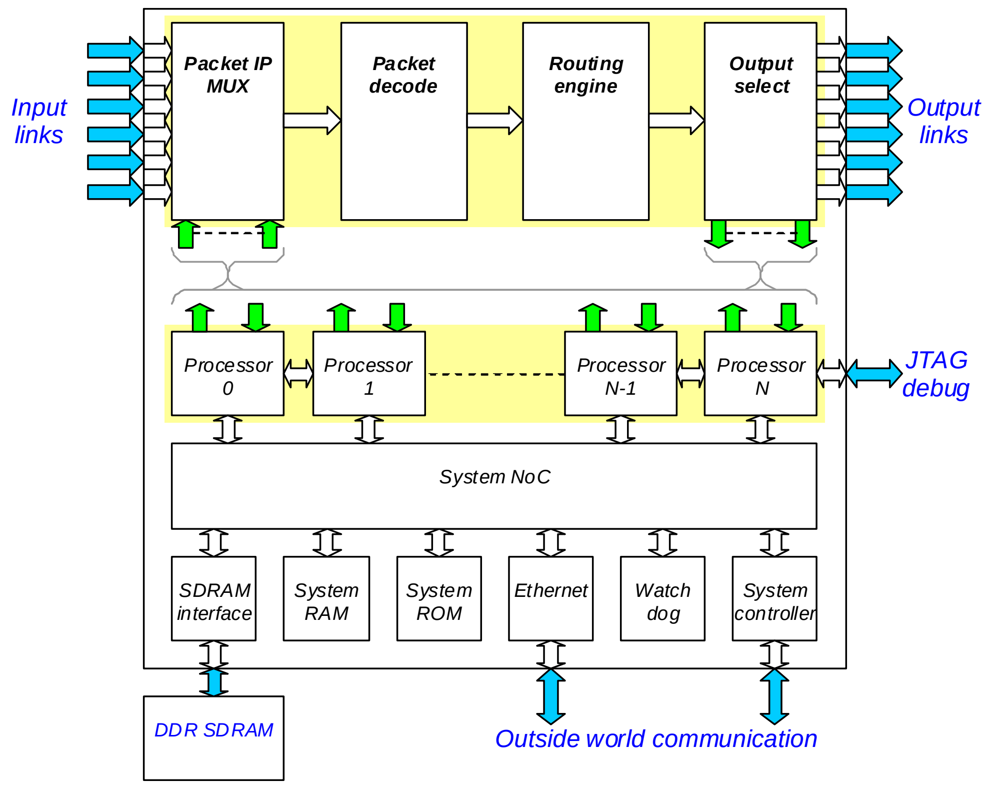

Figure 1: The internal architecture of a SpiNNaker node

rate of around 1 kHz.

We note in passing that the self-organising techniques described here open the door to an implementation of real-time plasticity, a long-standing drawback of most neural simulation systems. Whether this will be useful in practise remains to be seen; we estimate that the computational cost of deriving the routing tables dynamically to be comparable with a bootup.

Before moving on to the detailed body of the paper, we note that (a) the architecture is also extremely well suited to a large body of computational problems that have nothing to do with neural simulation, and (b) the problems of *obtaining* and manipulating a meaningful (neural) netlist of a billion elements are significant.

I. 2 SpiNNaker - what is it?

The internal structure of each SpiNNaker chip is shown in figure 1. The chip contains twenty ARM 9 cores, ostensibly identical, except that one of these cores is arbitrarily designated as a 'monitor' processor (the designation is soft and may change in real time). The monitor processor is intended to play no part in the simulation activities; it is there to provide 'local overseer' capabilities. The other processors are called 'fascicle processors'. Communication between cores is brokered by a packet-passing mechanism, described in detail in the next section. Note that cores on the same chip and different chips all communicate via the same mechanism - packets issued by a core go into the chip packet router, where they may be forwarded to cores on the same chip, any adjacent chip, or both.

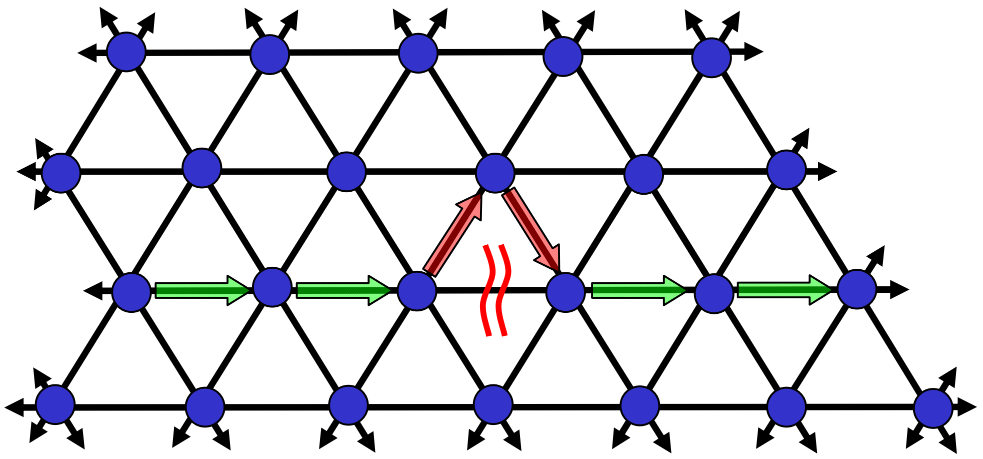

Figure 2: Low level link fault recovery

The six bi-directional links allow the 50,000 chips to be interconnected as a hexagonal mesh. Mesh edge effects are avoided by mapping the mesh onto the surface of a toroid, so that the environment as seen from any individual chip is isotropic and homogeneous, and all the chip environments are identical. The choice of a two-dimensional hexagonal mesh is pragmatic: it facilitates a rudimentary automatic low-level link fault recovery process (see figure 2), but aside from that, any connectivity is possible (limited, of course, by the constraint of six ports per chip). The infrastructure development described here does not assume or require any specific physical connectivity between the chips.

I. 3 The communication perspective

The chain of reasoning that drove the high-level design is derived from the quantitative characteristics of the biological neural systems it is designed to model. The human brain comprises in the region of 10^{11} neurons; the objective of the SpiNNaker work is to model 1% of this scale, which amounts to a billion neurons. Each neuron in the brain connects to thousands of other neurons. The mean firing rate of neurons is below 10 Hz, with the peak rate being 100s of Hz. These numerical points of reference can be summarized in the following deductions:

●	10^9 neurons with a mean fan in/out 10^3	==> 10^{12} synapses.
●	10^{12} synapses with ~4 bytes/synapse	==> 4.10^6 Mbytes total memory.
●	10^{12} synapses switching at ~10Hz mean	==> 10^{13} connections/s.
●	10^{13} connections/s, 10 instr/connection	==> 10^8 MIPS.
●	10^8 MIPS using ~100MHz ARM	==> 10^6 ARMs.

So 10^9 neurons need 10^6 ARMs, whence:

●	1 ARM at ~100MHz	==> 10^3 neurons.
●	1 node with 20 ARM968 + 80MB	==> 2.10^4 neurons.
●	5.10^4 nodes with 2.10^4 neurons/node	==> 10^9 neurons.

The above numbers all assume each neuron has 1,000 inputs. In practice this number will vary, and it is probably most useful to think of each ARM being able to model 1M synapses, so it can model 100 neurons each with 10,000 inputs, and so on.

The system will be inefficient unless there is some commonality across the inputs to the set of neurons modelled on a processor, so that each input event typically connects to tens or hundreds of neurons modelled by a processor. In biology connections tend to be sparse, so, for example, a processor could model 1,000 neurons, each of which connects to a random 10% of the 10,000 inputs that are routed to the processor. The standard model assumes sparse connectivity.

All the cores communicate via packets, and all the port connections are 1-to-1 - there are no busses. This is idealised in figure 3.

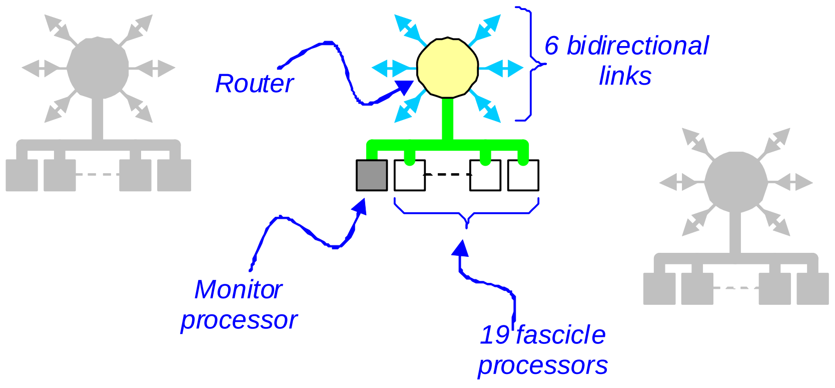

Figure 3: The communication perspective

II The communication infrastructure

II. 1 Requirements

The communication infrastructure must be scalable and fast. The latter requirement comes directly from the previous section, and the former from our ambition to push past the million core processor level at some point in the future.

A further problem that we have to acknowledge is that in a system of a million cores and 300,000 interchip links, faults and failures will occur, both dead on arrival and lifetime. Like the biological counterpart, the aim (hope?) is that the performance of the system will degrade gracefully as failures occur.

Biological neurons communicate via electrochemical impulses travelling along nerve fibres (action potentials). We idealise these to individual events, with a single element state vector, their launch time. Having made this idealisation, we push even further and discard the time, because the simulation system is operating in real time. We represent the neural action potential as a packet called **multicast** (MC). These packets are lightweight, and will provide the bulk of the system traffic during a simulation. For housekeeping, we provide two other packet types: **nearest neighbour** (NN) and **point-to-point** (P2P). The chip router can distinguish between these three types, and handles the packets accordingly.

What are these packet types, and why is there a communication infrastructure problem?

II. 2 Nearest neighbour packets

The nearest neighbour packet type is handled entirely by hardware, and requires no infrastructure programming; it is the bootstrap communication mechanism. It is intended to allow monitor processes to communicate - the receiving router will always direct the packet to its monitor processor, and the sending router has no interest in the exact origin of the packet (there is nothing to stop a fascicle processor sending a NN packet, but it hard to see the value of this). Packets may be addressed to a specific output port, or all six simultaneously. The internal structure of the packet is shown in figure 4.

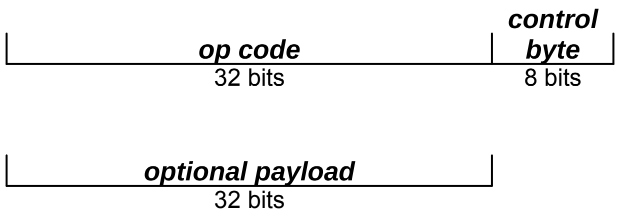

Figure 4: Internal structure of the nearest neighbour packet

II. 3 Point-to-point packets

The point-to-point packet internal structure is shown in figure 5. It allows monitor processors to communicate directly with each other, irrespective of where they lie in the machine. (These packets are the intended mechanism whereby a fascicle processor can communicate with its monitor. Communication in the opposite direction - monitor to fascicle processor - is achieved by the monitor writing to the shared chip memory, and then forcing an interrupt in the fascicle processor to go and look at the message.)

This brings us to our first communication infrastructure problem: packets in transit are passed from router to router via the inter-chip links. How does an individual router know what to do with a P2P packet? At the

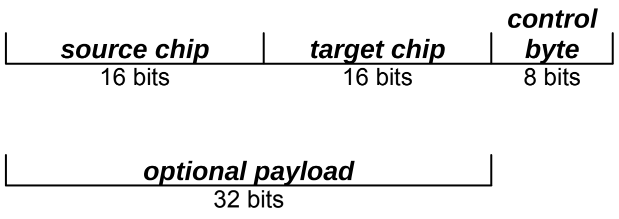

Figure 5: Internal structure of the point-to-point packet

hardware level, each router is provided with a 64k x 3 bit RAM, the contents of which dictate what the router does with a P2P packet: that packet may need to be forwarded (to which port?) or this chip may contain the destination monitor. This RAM needs to be defined.

II. 4 Multicast packets

The internal structure of the multicast packet is shown in figure 6. Like the P2P packets, these packets have a routing table, although this is somewhat more sophisticated than the P2P table. MC packets allow *neuron* to communicate with *neuron*. (Recall each *core* plays host to around 1000 neuron models, and the neural fan-

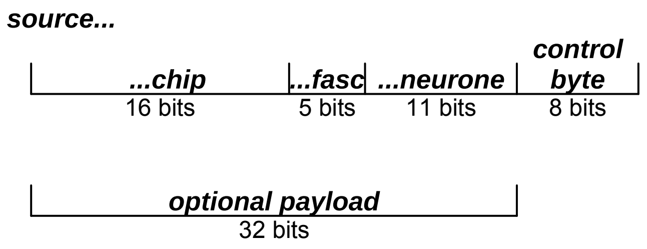

Figure 6: Internal structure of the neural event multicast packet

in/out for biological systems can vary from 1 to 10^5.) The MC routing table is capable of carrying a single event through the network until such time as it is necessary to physically split it; see figure 7.

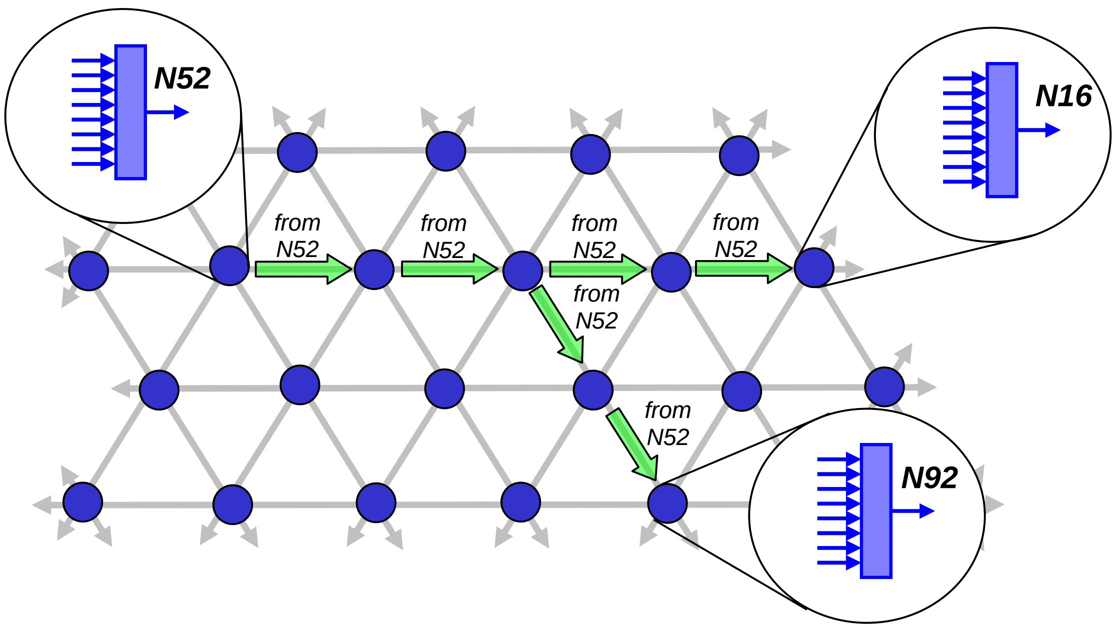

Figure 7: MC packets split by the node routing table

This brings us to the second and third infrastructure problems:

How do we decide which neuron model to map to which core, and, having done that, how do we derive the MC routing table entries?

III The need for self-organisation

All three problems are closely related to the electronic design automation Automatic Place & Route (APR) problem. This involves the physical placement and interconnection of components onto a silicon substrate or printed circuit board. The area first became important in the early 1980s, when circuit complexities became intractable to humans. It is generally considered to be a solved problem. Why, then, can we simply not borrow and adapt these techniques for use here? The answer, in general, is scale and real-time adaptability. Conventional APR requires a host machine to have a complete overview of the system it is solving. Where this is not feasible, hierarchical decomposition allows us to reason about parts of a circuit in isolation, and then stitch these solutions together. The loss of quality of solution as a consequence is not considered important in the silicon industry.

For the P2P route structure, we have 50,000 tables, each containing 64k entries (of which only 50k are used). Thus we need to compute 2.5 GByte of data (and then load it into the SpiNNaker machine). By the standards of today, this is far from impossible, but equally it is not a trivial task.

For the MC problem, we have to map 10^9 neurons to 10^6 cores, and (taking the average neural fan-in to be 10^4), establish 10^{13} neural interconnects, which then need to be loaded into 50,000 MC route tables. These numbers indicate we cannot even *hold* the data in a 32-bit offline machine; and any place/route mapping algorithm we use cannot be more than linear in both time and memory footprint.

Hierarchy could possibly deliver success for the 10^6 core machine, but probably not beyond.

Fortunately, we have an alternative: a million core parallel processing machine, and with careful choreography, we can actually generate the results where they need to be.

IV The P2P network structure

The SpiNNaker machine comes with a hardware NN packet communication system built in. We can use this to build up the P2P routine table entries dynamically. Further, the process is sufficiently computationally cheap that the tables can be redefined 'on the fly' if a fault occurs. (Note that, in general, a partial rebuild is not safe; it is rarely possible to guarantee the absence of loops.)

Preconditions: we assume

- Each chip has a unique ID.
- Each chip 'knows' it has a known, arbitrary number of named ports available.
- The entire system is configured as a simply connected random graph.

The 'knowledge horizon' of each chip is illustrated in figure 8. Our specific embodiment (6 links/chip, a regular hexagonal layout) is just one such system.

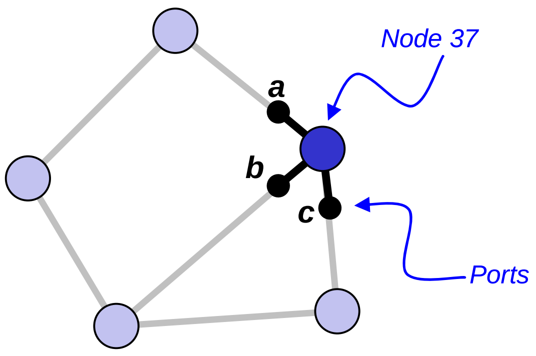

Figure 8: Power-up 'knowledge horizon' of a single node

IV. 1. The algorithm

Each node executes the following pseudocode:

```
Clear_P2P_route_table();
NN_send(myID,allPorts);
while (forever) {
  NN_recv(fromID,myIPport);
  if (route_table[fromID] != 0) continue;
  route_table[fromID] = myIPport;
  NN_send(fromID,all o/p ports except myIPport);
  if (routetable full) break;
}
Kill_outstanding_message_Q();
```

A series of snapshots depicting the execution of this code are shown in figure 9, for the route table entries for just two nodes, N2 and N6.

Each initial post by a node initiates a 'search wavefront' spreading out through the graph, with each node encountered by the ever-expanding wavefront storing the 'way back' to the source node. The process stops when all the nodes have all their tables full. The figure assumes that all the cores are executing in lockstep, for the sake of clarity - obviously this will not be the case in reality.

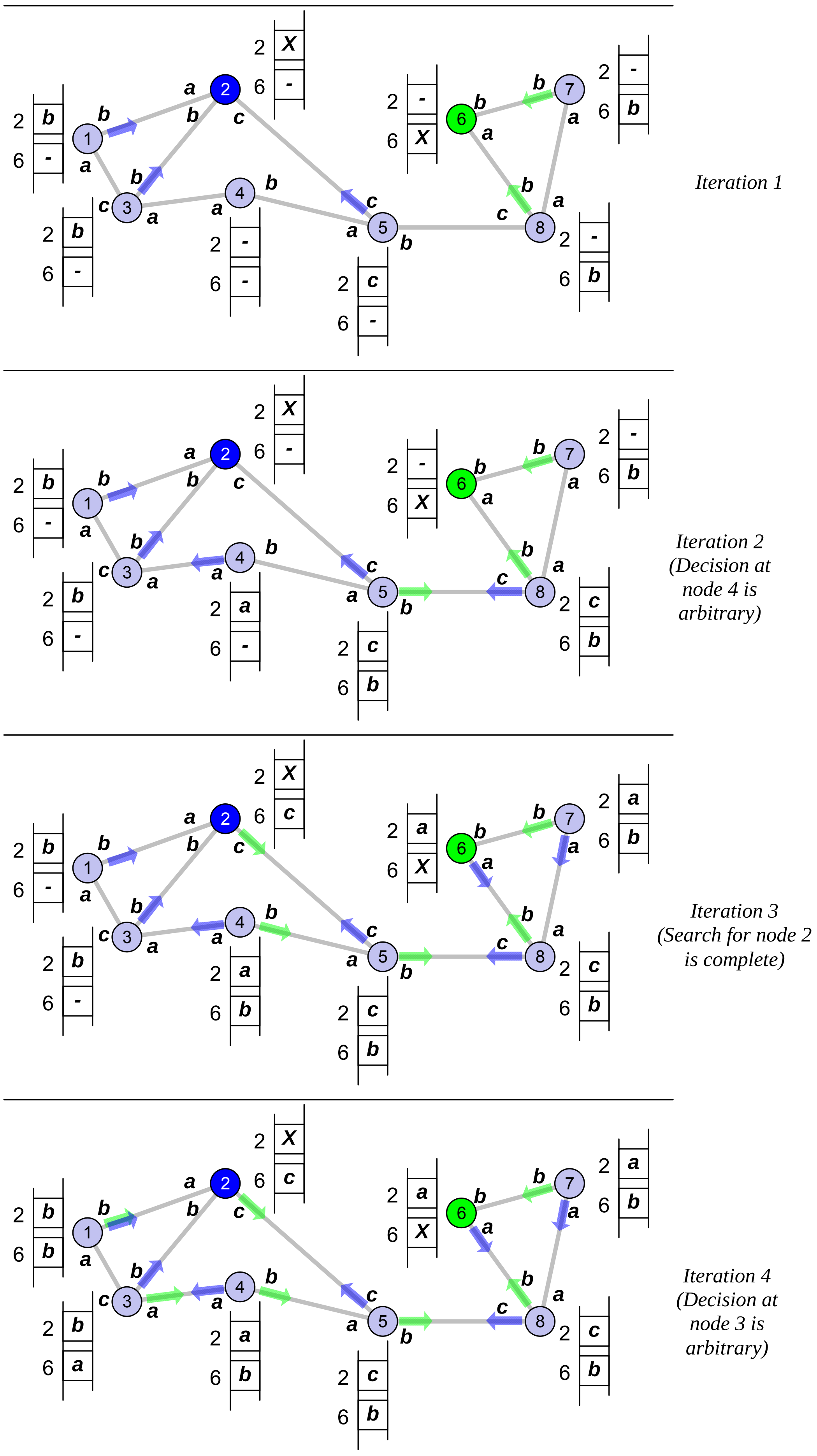
Iteration 1
Iteration 2 (Decision at node 4 is arbitrary)
Iteration 3 (Search for node 2 is complete)
Iteration 4 (Decision at node 3 is arbitrary)

Figure 9: Building up the P2P route table

IV. 2. Quality of solution

The processors in the SpiNNaker machine are not synchronised; an optimal (minimum hop length) route set will only be generated if the loops on each processor execute at the same rate. While this may be true initially, the forking control flow contained in the loop body means that the search wavefronts will propagate at different rates - the effect is illustrated in figure 10. Nominal isochrones show node 3 taking as long to complete as node 5 + node 6 + node 7 + node 8; consequently the table entry for node 5 gives a route with a hop length of 5 to get to node 1, where the obvious shortest route has a hop length of 4 (via nodes 4-3-2). However, the algorithm will always produce *a* correct answer, even if it is not necessarily the shortest route. In the worst case (when N nodes are strung together in a single line), the entire process will terminate in the time taken for the slowest processor to execute the loop N times.

V. The MC network structure

The MC routing structure is designed to facilitate the communication of 10^9 neuron models. This is achieved by distributing the definition of the neural interconnect network throughout the system (in the MC routing tables) so that the data carried by the MC packet can be as compact as possible. The packet itself contains the *source* address only; each individual route table knows what to do with packets emanating from that address

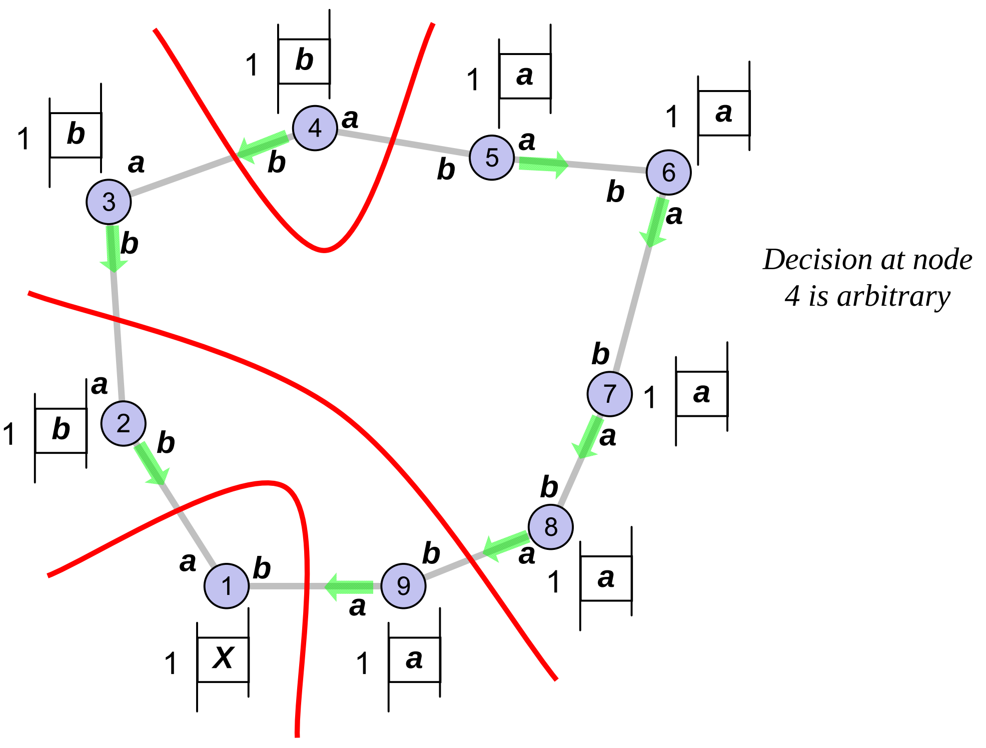

Figure 10: Timing effects in the construction of the P2P route tables

(i.e. forward to another port and/or deliver to a local fascicle processor), and the 'data' carried by the packet is, in effect, the existence of the packet itself.

V. 1 The MC routing table.

The structure of the MC route table is shown in figure 11. The 32-bit source key is input to a 1024 x 32 bit ternary content addressable (or associative) memory (CAM). As it is a ternary (0, 1, X) CAM, in general, multiple hits will be both possible and common. These hits are written to a 1024 x 1 bit hit register. All but the most significant single bit in this register are discarded, and this single remaining bit treated as a 1024 bit 1-hot and passed into an address encoder. This generates a 10 bit binary equivalent, which drives a 1024 x 26 bit lookup RAM. The 26 bit word so generated consists of a 6-bit nibble and a 20-bit nibble: the 6-bit nibble represents an n-hot external link indicator (0-5) to which the packet is forwarded (for example 010110 would

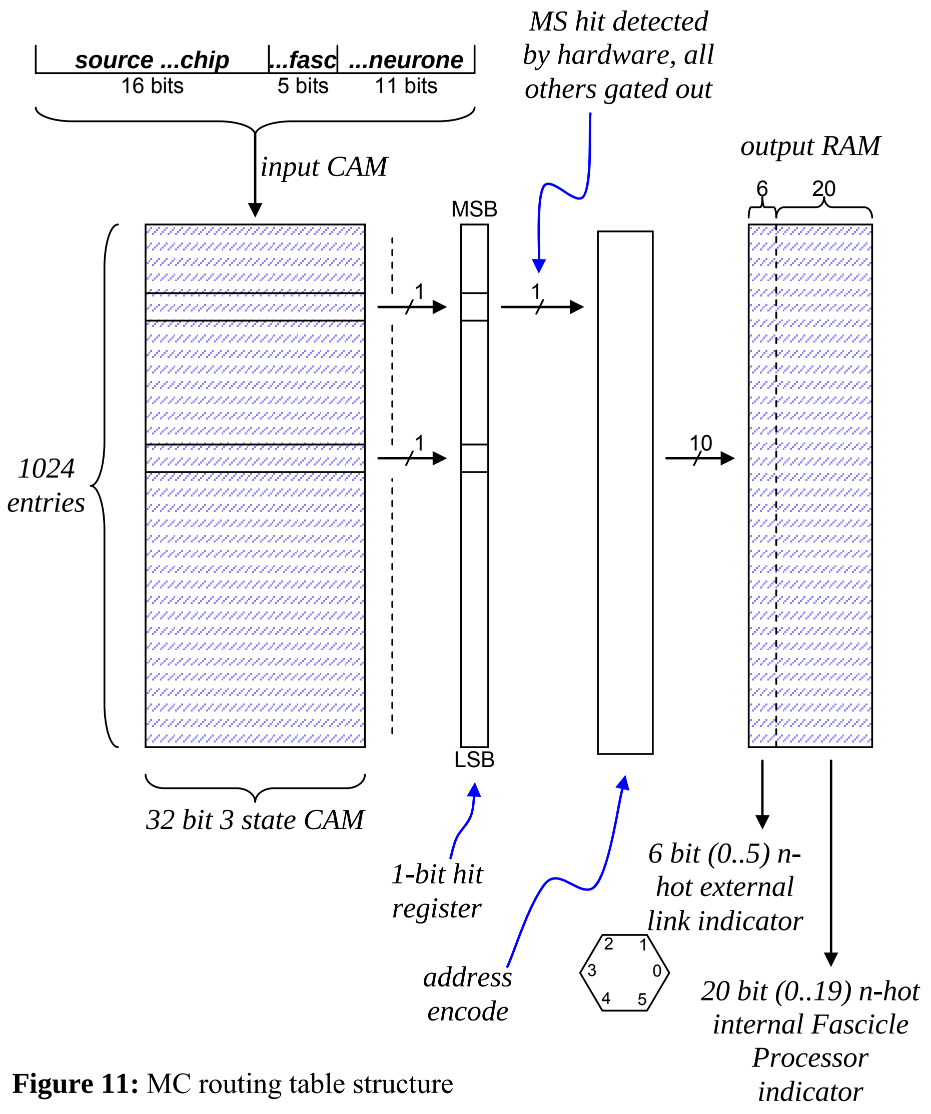

Figure 11: MC routing table structure

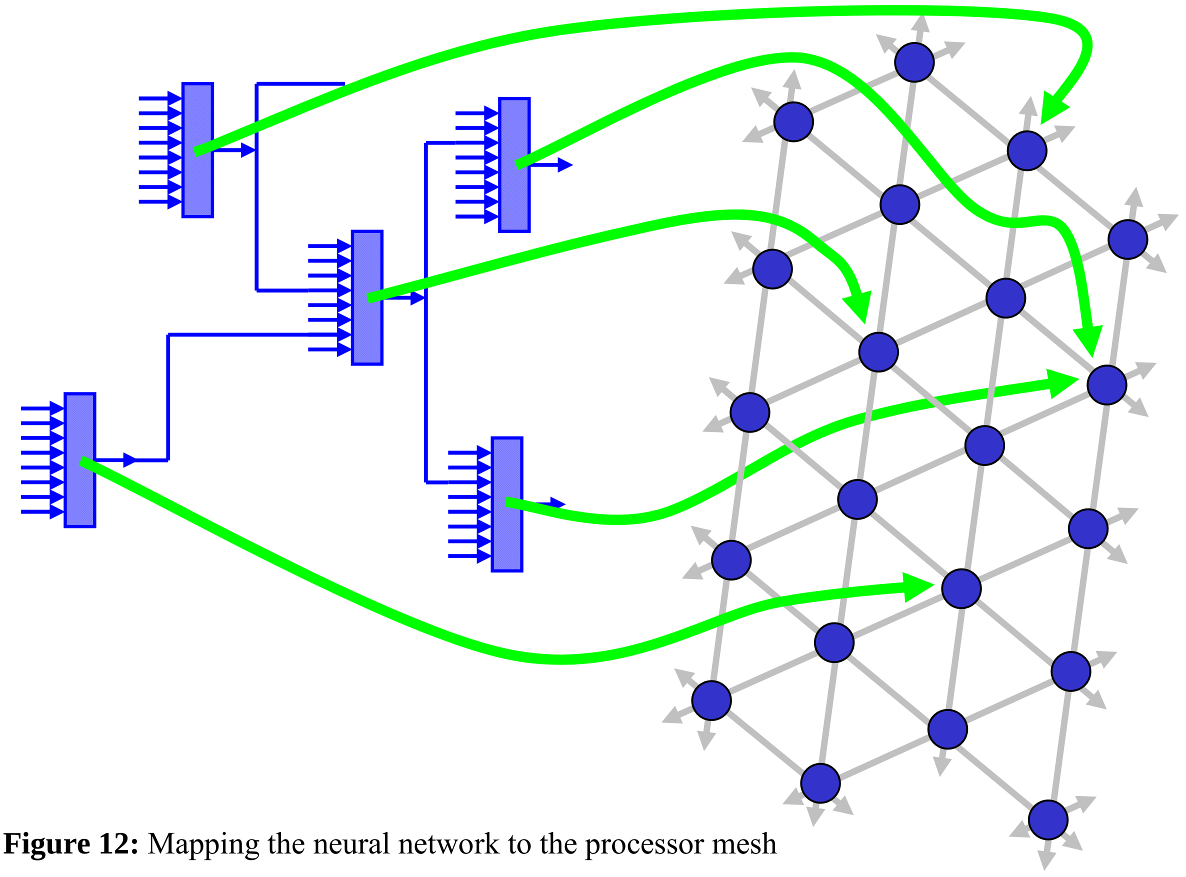

Figure 12: Mapping the neural network to the processor mesh

cause the packet to be routed to external links 1, 2 and 4), and the 20-bit nibble represents an n-hot internal fascicle processor address (0-19) to which the packet will be forwarded, triggering an interrupt as it arrives. (For example, 00001000100100000000 will cause packets to go to fascicle processors 8, 11 and 15 *on the current chip*. It is easy to see how packets may be duplicated by this mechanism.

Having described the physical embodiment of the route table, we have to address the issue of the origin of the RAM contents.

V. 2 Place and route

SpiNNaker - in its primary application space - is a distributed parallel simulation system, tuned to the real-time simulation of extremely large neural networks. Sidestepping the issue of where these network definitions may be found, we are faced with the problem of mapping a network of around 10^9 neurons onto a network of 10^6 processors - see figure 12. Conventional automatic place/route/mapping techniques (requiring significant hierarchical definition to make the problem size tractable) can, of course, be applied to this, but even so, this pre-processing cost will be significant. Task separation like this also renders dynamic circuit topology reconfiguration in real time a complete impossibility. Better, then, is to find some way of exploiting the power of the SpiNNaker machine itself to perform the mapping.

V. 2.1 Distributed placement on SpiNNaker

The SpiNNaker placement problem is similar to the problem of minimising the overall energy of a set of mutually repulsive bodies, subject to various constraints. A cloud of electrons *in vacuo*, for example, will simply fly apart. A group of electrons in a closed box will distribute themselves around the perimeter of the box. In a two-dimensional system, where the edges of the geometry identify such that the two-dimensional plane becomes a three-dimensional toroid, the configuration of minimum energy corresponds to the particles being distributed uniformly over the surface. It is this kind of solution we seek for the distributed placement system in SpiNNaker. Note that although our solution technique may be inspired by field theory, we are in no way attempting to solve a physical field problem, and the laws of physics may be perverted or ignored completely to suit our purposes.

The computational and algorithmic constraints include:

- The SpiNNaker nodes are interconnected to a few local neighbours only; as the overall node topology may not be regular and certainly will not be known to each node, the 'knowledge horizon' of each node is deemed to be its immediate neighbours.
- The n-body distribution problem in physics relies for its solution upon the notion of a conservative vector field. Whilst an arbitrary discrete network (of processors) can support the concept of a scalar field, and even distance -at a push (hop length) - the notion of field *slope* cannot be usefully defined
- Asymptotic solutions do not sit easily in distributed discrete networks - they can oscillate. Whilst this in itself is not a problem (we are attempting to solve a discrete placement/mapping problem, not a problem in field mechanics), the *detection* of this oscillation and the *realisation* that we have a solution is non-trivial.

In outline, the algorithm may be explained by considering the extremely simple system of figure 13. The SpiNNaker machine is configured as a single loop of ten, 2-port nodes, P0..P9. Without loss of generality, two bodies are initially randomly mapped to nodes P1 and P2. The goal is to get the system to self-organise, such that the bodies end up as far away from each other as possible, subject to the constraints/comments above.

The first step is to establish a 'field' throughout the entire processor mesh, consisting of the scalar sum of the contributions from each body in the system. This field

- must decrease monotonically with distance (hop count) from its source
- must be non-linear with hop length.

Such a field may be quickly established using the NN packet infrastructure. Calling the field due to A F_A and so on, figure 12 (the rows labelled 'iteration 0') show how the total field ($\sum F$) at each processing node may be built up.

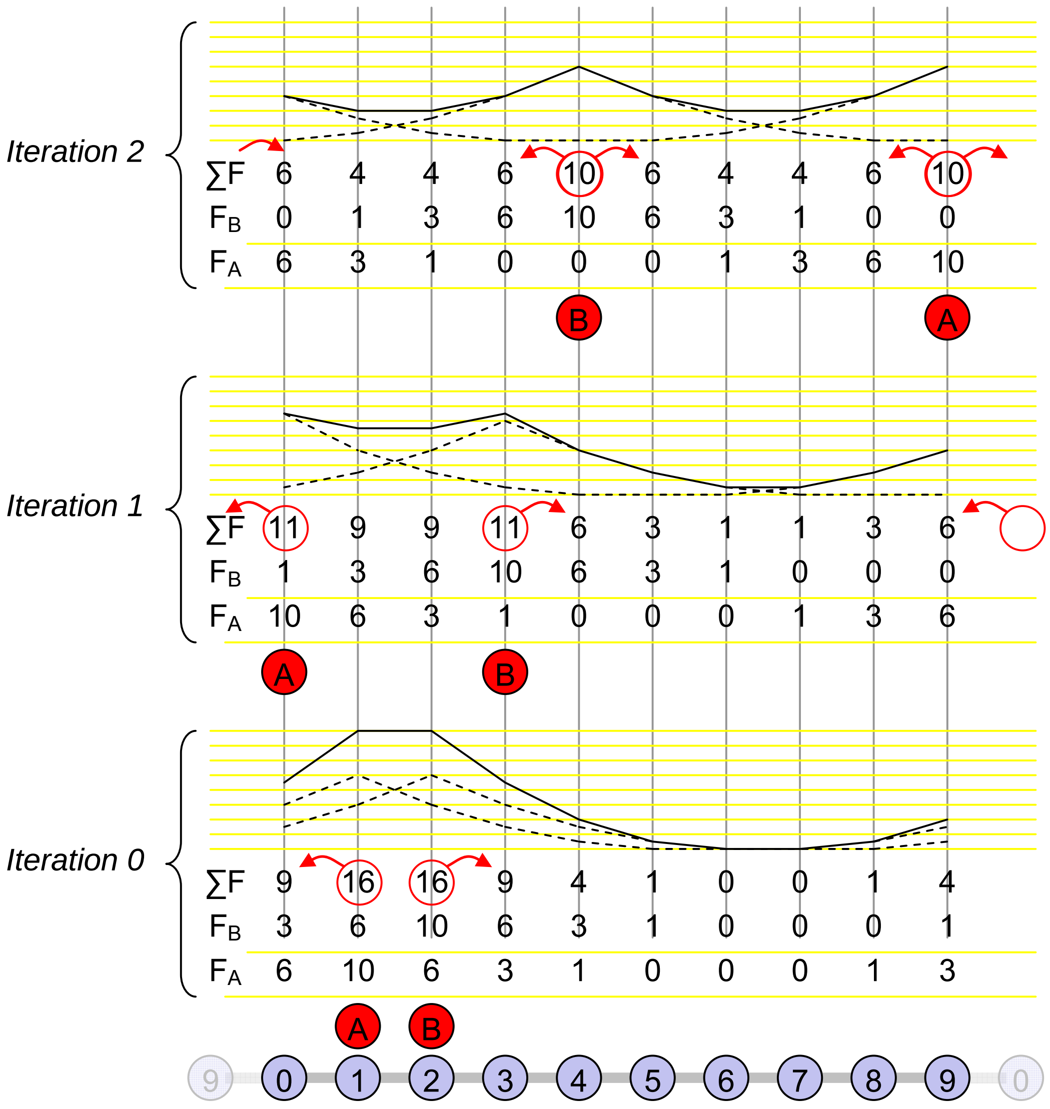

Figure 13: Self-organising placement

Looking now at each node in turn:

The lowest neighbour field value of node A(:processor P1) is 9(P0), and the lowest neighbour field value of node B(:processor P2) is also 9(P3). Thus we allow A and B to migrate to processors P0 and P3 respectively.

Repeating the process (iteration 1) shows that the lowest neighbour field value of node A(:processor P0) is 6(P9), and the lowest neighbour field value of node B(:processor P3) is 9(P4). Thus we allow A and B to migrate once more.

By iteration 2, the immediate neighbours of each node indicate no route by which they can reduce their ambient field value, and the system is stable.

Note that the algorithm

- only uses the scalar value of field values at adjacent nodes - there is no necessity for the notion of slope
- is appropriate for graphs of arbitrary topology: the migration path is simply the one with the lowest absolute field value.
  - If all paths have the same field value, the node does not migrate
  - If the lowest field value is shared between more than one path, the decision is resolved by resort to the *neural* topology.

The big advantages are (a) when the algorithm terminates, all the necessary data is actually resident in the processing node that will ultimately require it, and (b) all the placements are occurring simultaneously.

V. 2.2 MC routing

Once the neural model : SpiNNaker node mapping is in place, the establishment of the MC routes themselves come down to a question of realising the inflection points on each MC route; this may be achieved by a mechanism similar to that outlined in section IV.

VI. The way forward

The SpiNNaker machine is a major undertaking, from the perspectives of both hardware and software. Its massively parallel asynchronous nature will produce unique strengths for certain classes of computations, and the aspiration to model nervous system function has driven the fundamental hardware design decisions which, in turn, demand novel solutions to the problems of configuring and programming such a machine. We are now in unmapped territory, without the huge body of expertise that can be applied to conventional systems; however, we believe that taking inspiration from the self-organising principles of biological systems will guide us toward effective solutions in the future.

VII. References

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

[3] S.B. Furber and S. Temple, "Neural systems engineering", *J. R. Soc Interface*, **4**, no 13, pp 193-206 Apr 2007.

[4] A. Rast, S. Yang, M. Khan and S.B. Furber, "Virtual synaptic interconnect using an asynchronous network-on-chip", in *Proc 2008 int'l Joint Conf on Neural Networks (IJCNN'08).*

[5] X. Jin, S.B. 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 (IJCNN'08).*

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

[7] M.M. Khan, D.R. Lester, L.A. Plana, A. Rast, X. Jin, E. Painkras and S.B. Furber, "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor", in *Proc 2008 int'l Joint Conf on Neural Networks (IJCNN'08).*