SpiNNaker: Event-based simulation - quantitative behaviour

Andrew D. Brown, *Senior member, IEEE*, John E. Chad, Raihaan Kamarudin, Kier J. Dugan and Stephen B. Furber, *Fellow, IEEE*

Abstract—SpiNNaker (Spiking Neural Network Architecture) is a specialised computing engine, intended for real-time simulation of neural systems. It consists of a mesh of 240x240 nodes, each containing 18 ARM9 processors: over a million cores, communicating via a bespoke network. Ultimately, the machine will support the simulation of up to a billion neurons in real time, allowing simulation experiments to be taken to hitherto unattainable scales. The architecture achieves this by ignoring three of the axioms of computer design: the communication fabric is non-deterministic; there is no global core synchronisation, and the system state - held in distributed memory - is not coherent. Time models itself: there is no notion of computed simulation time - wallclock time is simulation time. Whilst these design decisions are orthogonal to conventional wisdom, they bring the engine behaviour closer to its intended simulation target - neural systems. We describe how SpiNNaker simulates large neural ensembles; we provide performance figures and outline some failure mechanisms. SpiNNaker simulation time scales 1:1 with wallclock time at least up to 9 million synaptic connections on a 768 core subsystem (~1400th of the full system) to accurately produce logically predicted results.

Index Terms—Event-based computing, neuromorphic computing, neural system simulation, real-time simulation, specialised simulation platforms

1 Introduction

GENERAL-purpose (low physical thread count) computers have been used for simulation applications since they were first invented. As with most technologies, the capabilities of the underlying machines have advanced more-or-less hand-in-hand with the expectations of users. However, general-purpose computers are, by definition, designed in the absence of knowledge of their intended application (which makes them ill-suited for almost every *specific* task), and the plummeting cost of hardware has allowed the rise of bespoke engines, tailored to specific (types of) computing tasks. Neural simulation is an application where the computing resource necessary to undertake simulations of the scale necessary to demonstrate emergent behaviour is far outstripping the capabilities of commodity machines, and severely testing those of multi-million dollar supercomputers. Other application areas in this class include large-scale particle/particle and particle/field problems (computational chemistry, cosmology, high-energy theoretical physics), weather modelling, financial market stress testing, and others[30]. This paper focuses on the performance of a machine specifically designed to simulate the behaviour of extremely large numbers of small data packets moving through an extremely large graph *in real time*: a mammalian nervous system.

Neuromorphic computing is a branch of computer science focused on computing engines specifically designed to simulate the behaviour of large (many millions) aggregates of neurons. The paper describes

- The SpiNNaker simulation engine, its architecture, programming model and specific configuration challenges.
- The specific algorithmic and numerical techniques employed to predict the behaviour of the individual neurons.
- A set of performance benchmarks, showing how the system behaves.

These benchmarks are also analysed on BRIAN[1] and a naive, simple neural simulator called ANSWER, and quantitative performance outcomes compared. Both BRIAN and ANSWER execute on conventional single-thread architectures.

The benchmarks are artificial neural network constructs with logically determined correct behaviours. They consist of sets of idealised synfire ring oscillators [2,3], beating with each other. They are not chosen for significant biological relevance or fidelity (although they are sympathetic to the types of problem that are neurologically realistic), but to stretch and explore the capabilities of the three simulators in terms of simulation ability. The focus of the paper is on simulator *performance*, both in terms of speed and resource consumption.

In the context of this paper, "design intent" indicates that the system is operated in the manner in which it was intended. If the user steps outside this perimeter, the sys-

- *A.D. Brown is with the Department of Electronics & Computer Science, University of Southampton, SO17 1BJ, UK. E-mail: adb@ecs.soton.ac.uk.*
- *J.E.Chad is with the Department of Biological Sciences, University of Southampton, SO17 1BJ, UK. E-mail: j.e.chad@soton ac.uk.*
- *R. Kamarudin is with the Department of Electronics & Computer Science, University of Southampton, SO17 1BJ, UK E-mail: mrbk1e14@soton.ac.uk.*
- *K.J. Dugan is with the Department of Electronics & Computer Science, University of Southampton, SO17 1BJ, UK. E_mail: kjd1v07@ecs.soton.ac.uk.*
- *S.B. Furber is with the School of Computer Science, University of Manchester, M13 9PL, UK, E-mail: steve.furber@manchester.ac.uk.*

tem - quite rightly - will not function correctly.

2 Specialised Simulation Platforms

SpiNNaker is certainly not the only computing engine to be utilised for the simulation of large neural systems:

2.1 BrainscaleS

The FACETS and BrainscaleS projects have developed a 'wafer-scale' system for neural simulation [4], using an analogue implementation of the adaptive exponential Leaky Integrate and Fire (LIF) neuron model (a two-variable, single compartment model)[5]. Since electronic component time constants can be more rapid than biological membrane time constants, the FACETS hardware operates at around 10,000 times notional biological real-time. The system comprises up to 384 analogue cores on a wafer (each simulating 512 neurons), which are interconnected via a custom packet-switched network to provide up to O(10^7) neuron models per wafer. Larger machines can be created by connecting together multiple wafers, via a secondary network that uses time-stamped event packets [6]. The accelerated time at which the FACETS system runs makes it a challenge for live interaction with the world (at least directly) [7], but is clearly valuable in terms of examining parameter sensitivities and exploring parameter spaces. Since the neural model is implemented in silicon, limited scope exists for modifying the behaviour of a neuron in the simulated network; this loss of flexibility is traded off against the energy consumption (typically 10^{-9} J/synaptic event - compare with conventional simulation technology on supercomputer clusters: 10^{0}-10^{-4} J/event and nature: typically 10^{-14} J/event [25][1]).

2.2 Cauwenberghs Integrate-and-Fire Array Transceiver (IFAT)

The IFAT[8] is a mixed-signal approach to neural simulation hardware, combining analogue neuron models with a digital synaptic interconnect. The neuron chips use a conductance-based LIF model, while the synapses are implemented by an event-based system in an FPGA that connects together multiple IFAT chips. Early versions used 2400 neurons/chip and up to 4M synaptic connections, and were shown to operate in real time in conjunction with a silicon retina for related visual processing tasks[9]. A later incarnation of the same basic plan [10] supports 2^{16} (64k) neurons per board, comprising four 16k neuron chips and a hierarchical communication system (again, using FPGAs); the overall system suggests that it could scale to four boards (256k neurons). The communication system capitalises on the all-or-nothing nature of the neural spike: its nature as a pure asynchronous event allows it to be conveyed efficiently in digital form using address event representation (AER)[11]. In AER systems each neuron is given a unique numerical identifier or 'address', and it is this address (the *source* address) that is transmitted to other neurons whenever the neuron spikes.

2.3 The IBM TrueNorth chip

TrueNorth [12] is a silicon implementation of a neural network. The system implements 4096 neurosynaptic cores, each of which models 256 neurons with 256 synaptic inputs. The chip uses 5.4 billion transistors but runs at only 70 mW, using an event-driven hardware style to minimise activity (and thereby save power - TrueNorth dissipates around 2.5×10^{-11}J/synaptic event). As with BrainscaleS, the neuron model is fixed (though here it is digital) with a per-input strength modulation shared across the 256 neurons in a core. The system is deterministic, allowing an exact software model to be used for development and debugging.

2.4 Other physical systems

The Stanford Neurogrid [13], which employs subthreshold analogue circuits for real-time performance, and the Cambridge Bluehive system [14], which uses digital circuits on FPGAs to deliver real-time performance are both powerful neuromorphic computing systems. The latter grew from a comparison project with SpiNNaker itself. A more detailed quantitative comparison of these (and other) architectures may be found in [33].

2.5 A broader perspective

The study of neural and cognitive systems is (rightly) vast, and this paper is not the place for even the briefest precis. This notwithstanding, it is observed that neuromorphic machines are but one tool in a large and growing panoply of systems dedicated to their elucidation There is much more to the (understanding of) the workings of the mammalian brain than the I/O relationships of the constituent components. To quote from [31]: "*In these approaches [the types of mechanistic simulation system outlined above] the brain is considered solely as a physical substrate. By analogy, this would amount to restricting the study of a computer to the description of its electronic circuits, or hardware, ignoring its software level that expresses algorithms under the form of programs.*" There are strong analogies between the theories outlined in [31] (and others), and the approach we have taken in SpiNNaker. Indeed, SpiNNaker is an ideal platform for the realisation of these concepts. [32] discusses the features of a massively asynchronous, parallel brain in ways, again, that resonate with the design principles of SpiNNaker. The paper closes with the words "*The brain may, in fact, be much more like asynchronous computers than synchronous ones. Or rather, asynchronous, parallel computers may end up being much more like the brain than parallel, synchronous ones.*"

2.6 Summary

The landscape of neural-specific simulation hardware is growing at a remarkable rate and is being explored in many dimensions, not all using bespoke hardware: The Blue Brain project[15] employs a conventional supercomputer, allowing for largely heterogeneous models (with

[1] This figure can be derived in a number of ways. *Top-down:* the human brain dissipates around 20W, contains around 10^{15} synapses and the neurons fire at around 1Hz, giving **2.10^{-14} J/event**. *Bottom-up:* A single ATP molecule can deliver around 10^{-19} J[27], and a synapse uses around 10^{5} molecules/transmission[26], giving **10^{-14} J/event**. Breaking this down even further, a neuron spiking requires about 10^{9} ATP molecules, so the power dissipated by the neurons is around 10^{-19} x 10^{9} x $10^{11} \approx$ 10W, and the corresponding figure for synaptic power dissipation is 10^{-19} x 10^{5} x $10^{15} \approx$ 10W; making a total power budget of 20W.

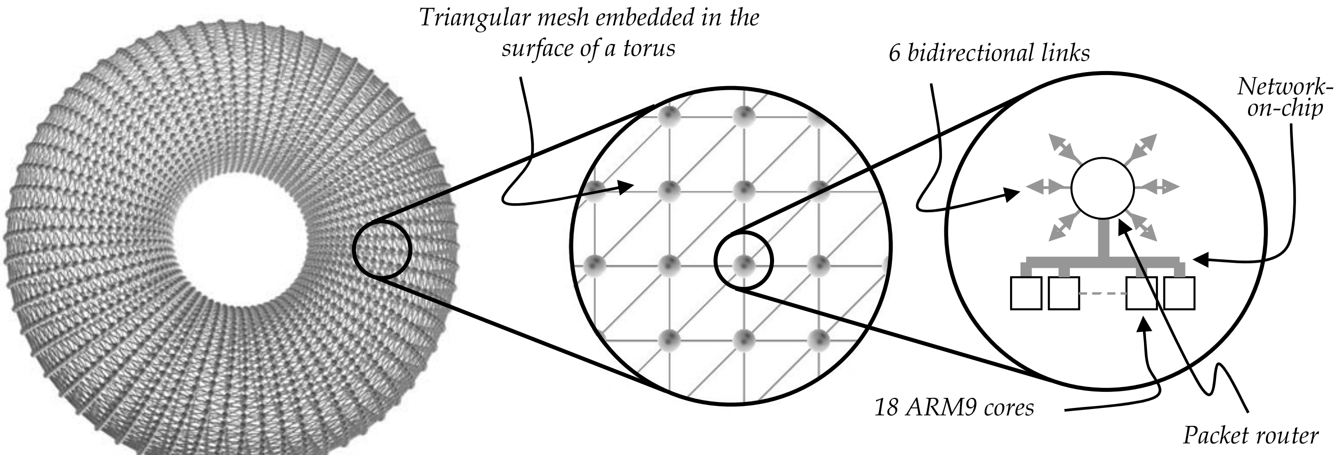

Fig 1: SpiNNaker architecture (graphics produced in part by the tool [25])

much detail, but with great energy cost). Conversely, engines such as those enumerated above use more efficient analogue electronic circuitry but comprise a single neural model with only limited flexibility.

SpiNNaker connects a very large number of general-purpose (digital) processors with an efficient custom network, providing the user extensive freedom to manipulate the specific details of the neural models employed. This flexibility underpins the power of the system: it exploits hardware for speed, and software for user configurability in a way not possible with a conventional or otherwise fixed architecture.

3 The SpiNNaker engine

SpiNNaker is composed of general purpose computing cores (ARM9), and so, in principle, can be made to do anything a programmer cares to define. However, the nature of the resource distribution (memory) and communication infrastructure make it ideally suited to any problem that can be cast into the form of a large number of interacting entities, each possessing a small but independent state - such as a neural network.

3.1 Machine architecture

The machine architecture is described in detail in [16-18], and outlined in fig 1. Starting from the left, at the highest level of granularity, the system is composed of a set of triangularly connected **nodes**, conveniently mapped onto the surface of a three-dimensional toroid. The nodes are identical, and the mesh is isotropic, which means that the physical size of the system is completely scalable from (in principle) 1, up to a hard limit of 256^2 nodes. (This limitation derives from the internal labelling of the nodes which allows only a 16 bit identifier.) Each node possesses

- An Ethernet connection, which supports communication with the outside world.
- A set of six bi-directional communications links, plus an associated routing subsystem.
- An internal network-on-chip, connecting the node links to a set of eighteen ARM9 cores.
- Various memory subspaces (see below), timers and counters.

1) Cores

Electrically, the eighteen ARM9 cores on each node are identical, but during initialisation, a (deliberate) hardware race occurs, achieving two things: one core (the **monitor**) is elected to oversee the behaviour of the node (and plays no direct part in the simulations), and sixteen (of the remaining seventeen) are elected to support the simulation capability. These are called **application** cores. (In a system of a million cores, it is unrealistic to expect that everything will be - and remain - perfectly functional. Fault-tolerance is built into the architecture at several levels [19].)

2) Memory maps

All the resources available to every core are memory mapped. Each core has a local 32-bit address space, containing 32Kbytes of private instruction tightly-coupled memory (ITCM), and 64Kbytes of private data tightly-coupled memory (DTCM), giving the cores a Harvard architecture. In addition, mapped onto the address space, is 128Mbyte of node-local SDRAM and 32Kbytes of SRAM. These latter tranches of physical memory support a fast way for cores *on the same node* to communicate with each other. In total, the million core machine has just over 7Tbytes of memory distributed throughout it.

3) Communication

The communication infrastructure built into the machine [17,18] supports the fast transmission of *packets* of data between individual cores. Unlike conventional multi core/process computer architectures, where the message size and choreography is usually defined by the user, in SpiNNaker, the packets are just 40 or 72 bits. This size restriction allows a packet to easily define the existence data of a neural spike, whilst at the same time making it possible for the routing infrastructure to be *entirely hardware*. This, in turn, makes the packet propagation throughout the system extremely fast: packets are transmitted between cores in a sequence of node hops (figs 1 and 6) - each hop taking around 0.2 us. Using a toroidal nodal interconnection topology gives the overall system a bisection bandwidth of around 5 billion packets/s. Packet trajectories may be 1 core to 1 core, or 1-core to many-cores: any multicasting is handled transparently by hardware. The packet interconnection graph (neural topology) is embedded in routing tables during the system configuration phase (see below), which permits every aspect of packet propagation throughout the entire system to be hardware brokered.

3.2 Programming model

There is no central overseer process; there is no overall memory coherency, there is no global clock. Like the mammalian brain, local computation occurs due to the responses of a neuron to the arrival of packet input (synaptic activity) - it is *interrupt-driven*, and the arrival of a packet is an *event:*

- A packet arrives (via hardware) at its destination core.
- The hardware loads the core program counter with the address of the appropriate *packet handler routine*, held in the ITCM.
- Code executes in response to the arrival of the packet. (This code has access to any *local* state data (held in the core DTCM or the node SDRAM/SRAM), plus the 40/72 bits of packet data.)

During execution, the packet handler routine may/may not launch packets of its own. *After* execution, the handler terminates and the core enters sleep mode. There is no "return to operating system", because there is no operating system in the conventional sense of the term. The core is woken (by hardware) in response to the arrival of a packet-induced interrupt; the packet is handled, and the core returns to quiescence. *The design intention is that most cores spend most of their time asleep.*

3.3 Neural model

One of the most challenging compromises for the designer of simulation models is always the level of detail of the model [25]. Fine detail produces high runtimes, and one of the inviolable axioms of SpiNNaker is real-time performance. An important use of simulation is to expose emergent behaviour of a system, and this can be a strong function of a host of model parameters. In wet biology, cells can be *spontaneously* active and that activity is modulated by the synaptic inputs. Whilst all the simulation systems used in this study can support this behaviour, it has been deliberately suppressed to make the quantitative comparisons presented in section 5 easier to interpret.

3.4 Time

The cores in SpiNNaker can run at a nominal 200MHz, but there is no attempt to phase- or frequency-lock these across the machine fabric. (Given the size of the machine, any attempt to do this would have more than doubled the design and implementation costs of SpiNNaker.) However, each node possesses two independent counter/timer units [20], providing $\boldsymbol{O}$(ms) signals to indicate the passage of wallclock (*biological*) time. These timing signals are incident on all the cores (just as packets can be), and awake their own handlers, allowing each core to update a local biological time value. (All handlers are expected to be very small - a few hundred machine instructions at most - so that *most* interrupts are handled immediately on arrival.)

3.5 Configuration

Configuration is a non-trivial pre-processing stage that involves taking the definition of the neural circuit (in terms of the interconnect topology) and generating from this the routing table entries and memory index structures located on each node. (Details of the process are described in [18].) The principal route into the tool chain is via PyNN descriptions, but we also have a domain-agnostic input channel that allows interconnect descriptions in a variety of formats. Viewing both the neural networks and the node topology as graphs, the functionality of the configuration software is to create a *1:many* mapping of *cores:neurons* (fig 2), turn this mapping into the appropriate configuration data and upload it (via the Ethernet) to the node mesh; whereupon it is distributed to the target nodes. The design intent is that each core should be capable of hosting around a thousand neurons; thus the largest possible SpiNNaker engine (256^2 x 16 application cores) should be capable of handling simulations of over a billion neurons, or around 1% of the human brain.

4 Neural simulation

We have now arrived at the point where the neural network (the topology of which is defined externally by the user) is loaded into the SpiNNaker engine. The topology of the neural graph is embodied in the routing tables distributed throughout the engine, and the functionality and state of the individual neurons localised in individual cores. The *routing infrastructure* (route tables, node network-on-chip systems, physical node-node interconnect) allows an individual neuron (the state of which is localised in a specific area of memory on a specific host core) to send and receive packets to and from other, specific neurons, *without any consideration or concern about how - or when - anything gets to its intended target(s)*. The wallclock time taken for a packet to travel from its source to its destination varies according to the placement (fig 2) derived by the configuration software; it will lie in the range 0.1us (for a pair of neurons mapped to cores in the same node), to $\sqrt{(2^{16})}/2 * 0.2 \sim 25$us for a pair of neurons mapped to cores on opposite sides of the largest possible SpiNNaker toroid. **The real-time packet transit time is a function of**

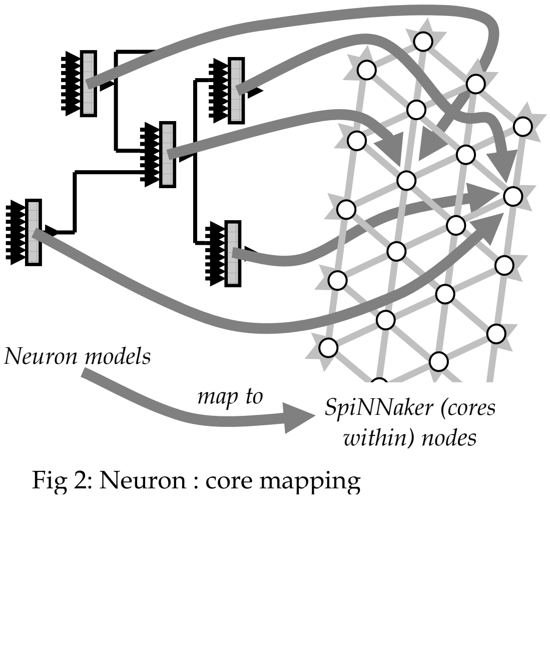

Fig 2: Neuron : core mapping

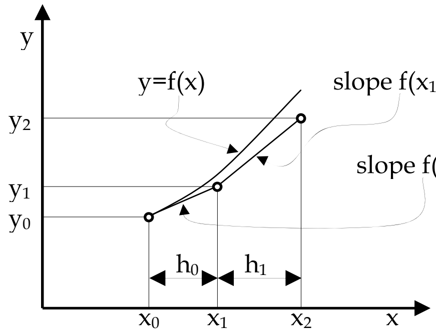

Fig 3: Conventional integration

the behaviour of the configuration software, not the biological topology of the network under simulation, but importantly, both these times are effectively *instantaneous* from a biological perspective. This range of variance (0.5us - 25 us) for packet transit time does not matter for the correctness of simulations attempting to discern emergent behaviours ay longer timescales [34].

The *functionality* of each neuron is embodied in the code associated with a packet interrupt, running on the appropriate application core. When a biological neuron receives an incident synaptic input, due to a pre-synaptic action potential, it reacts, and the nature of this reaction depends upon the type of neuron (i.e. the functionality) and its past excitation history (i.e. the state). In the broadest possible sense, most biological neurons exhibit some kind of integrate-and-fire behaviour, which can be captured in numerous ways. One of the most pervasive models (and one which can be used by SpiNNaker) is the Izhikevich model [21,22,25], which is a differential equation, the integration of which produces a wide range of biologically plausible behaviours. However, the exact form of the neural equations is irrelevant in the context of this paper, and the LIF model [25] is conceptually simpler, and will be used hereafter.

4.1 March-in-time (conventional) integration

In a conventional (single-thread, single processor) simulation, the differential equations associated with neural behaviour are integrated by some conventional method: Euler, Runga-Kutta, Adams-Bashforth [23]. Fig 3 depicts a conventional numerical solution in progress (with the errors exaggerated). The key point is that from the perspective of the software, simulation time is just another variable, to be manipulated alongside any other. The calculation proceeds at whatever rate is dictated by the host machine speed. (Also, we note that the timestep, ***h*** in fig 3, may not necessarily be a constant throughout the simulation, although it often is in most biological simulations, because the equations used are not usually numerically stiff[2])

4.2 Discrete time (event-based) integration

Consider a single LIF neuron, that has two synaptic inputs and a single output; incident upon the inputs are the pulse trains shown in fig 4 (*InputA, InputB*). The internal state variable (integrand) within the neuron might reasonably resemble the trace *State*. When excited, *State* is increased; when left alone, *State* decays (approximately) exponentially with the passage of (simulated) time. When *State* exceeds some internal threshold, the neuron fires (*Output*), and emits an output pulse (**action potential**) of its own. A consequence of the neuron firing is that it enters the **refractory period**, where all inputs in this time window are ignored. In SpiNNaker (fig 5) the explicit handling of the inputs is somewhat different. This is described in detail in the next section. Note that in fig 4, the abscissa is calibrated in *simulated* time. The relationship between simulated time and wallclock time on a conventional (single- or multi-core) architecture is not necessarily (indeed, is not usually) linear, or of much interest to the user, except when it becomes unacceptably large, and compromises user interaction.

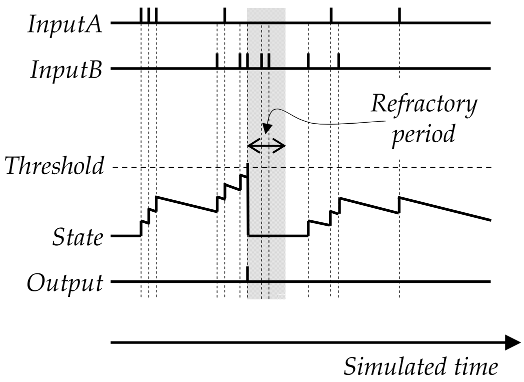

Fig 4: Continuous (event-based) integration

4.3 Real-time event-based integration (SpiNNaker) - in principle

With SpiNNaker, every neuron is represented by a discrete core, which spends most of its time quiescent. It is awoken by an incoming event, which can be from an incident neural event, or from the (real-time biological) clock. The corresponding waveforms are shown in fig 5. This is a digital system; state values can only *change* as a consequence of *events*. Thus the State (in fig 5) increases when an input is incident to the model, and decreases when a timer tick arrives. Between these occurrences, it is - has to be - constant. The pseudo-code (interrupt handlers) that generate these waveforms are shown below:

```
void OnPacket(MC_t *)
{
if (++state<=threshold)
  return;
SendPulse();
state = 0;
```

```
void OnTimer(TT_t)
{
state *= 0.95;
}
```

Each time a neural spike is incident on the neuron, the code `OnPacket()` is executed; each time a biological clock tick arrives, `OnTimer()` is executed. Note the abscissa in fig 5 is calibrated in *wallclock* (real) time.

If an event arrives whilst a core is still executing a handler for a previous event, the newer packet is queued until the core is free. As long as the wallclock *rate* at which incident

[2] In the discipline of electronic design automation (which is one of the heaviest users of numerical integration) a differential equation is considered *stiff* if the ratio of any pair of differential coefficients is > 10^{10}.

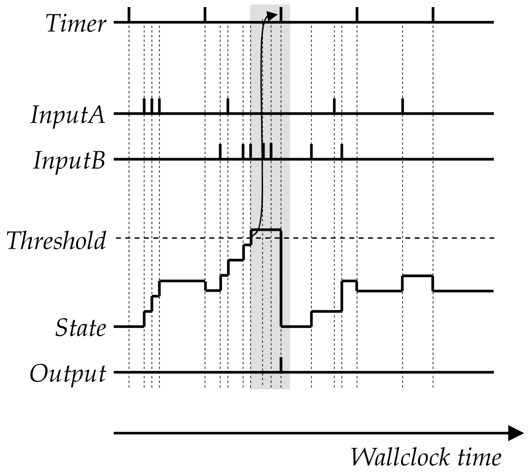

Fig 5: SpiNNaker integration

packets are consumed (handled) is higher than the (wall-clock) rate at which they arrive, the system operates successfully. The routing subsystem can buffer around a dozen packets, and even if some are forced into the next `OnTimer` slot, the *overall* accuracy of the simulation is not unduly compromised.

This may be counter-intuitive to an audience with a conventional computation background, where the order of events is generally of high importance. However, if the operation(s) carried out on the events as they arrive is simply integration - as it is here - then the temporal order of the integrands is unimportant. The consequence of the situation described here (that the integration is delayed: an unbiological processing artefact) is that (for a very small minority of cases) a generated spike may be delayed for a millisecond. Real biological systems are generally robust to this level of noise [34]. Subsequent sections here (4.4. et seq) expand on this.

4.4 Real-time event-based integration (SpiNNaker) - in practice

The outline explanation of the previous section illuminates the broad principles of the operation of SpiNNaker, but omits one important aspect: the goal of the SpiNNaker machine is to model the behaviour of a large number of neural models *in real time*, which it achieves by exploiting the massive parallelism of the routing infrastructure and the million cores contained within it. The system outlined in section 4.3 will integrate the neural equations as fast as it can (the ARM9 can run at 200MHz). We need to capture the *temporal* behaviour of the (biological) system under simulation, as well as the *functional* behaviour if the simulation is to contain any useful fidelity. Fig 6a depicts a simple biological system - of interest is the time history of neuron A exciting neuron B.

- Δt_{12} is the axonal delay of an impulse launched from neuron A. It is $\boldsymbol{O}$(ms), and a function of the biological geometry of the system under simulation.
- Δt_{23} represents the time taken for neuron B to react (via some suitable synapse) to the excitation from neuron A. It too is $\boldsymbol{O}$(ms), and is a function of biology and the state (history) of neuron B.

The corresponding scenario as implemented in SpiNNaker is shown in fig 6b. We assume (see section 3.2 and fig 2) that neuron A is mapped to SpiNNaker node N1, B to N4 and the intermediate routing tables configured such that A communicates with B via the routers on N1, N2, N3 and N4.

- The axonal delay (fig 6b) stored as a parameter in

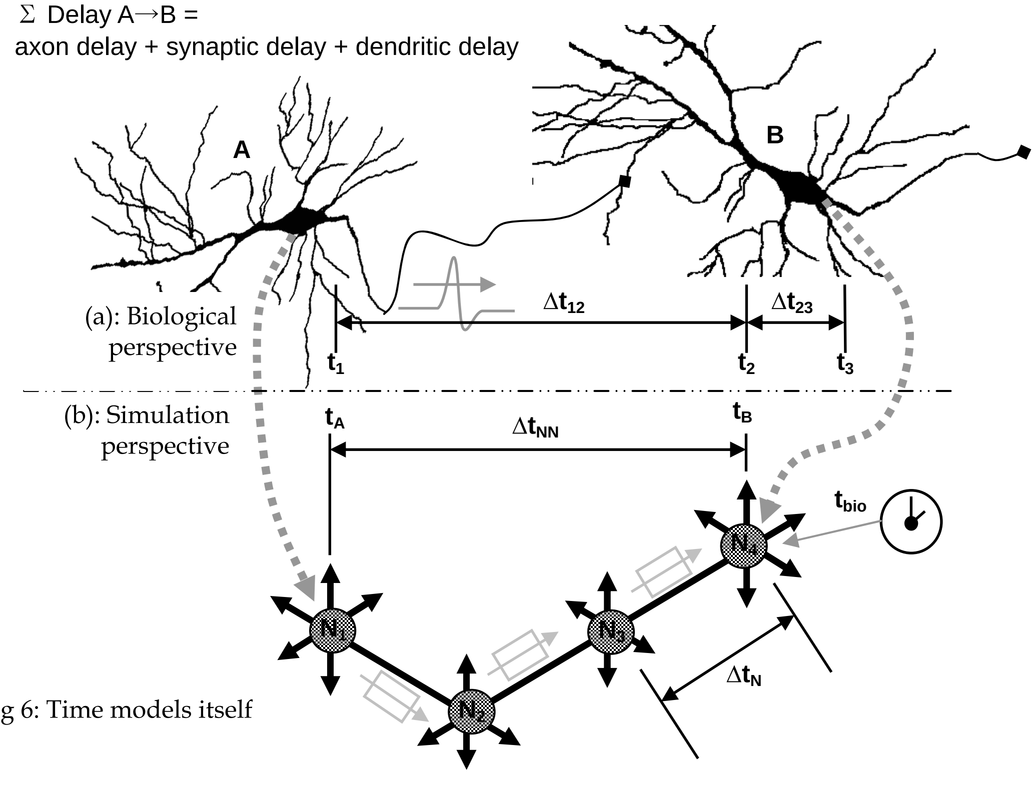
Σ Delay A→B =
axon delay + synaptic delay + dendritic delay
A
B
Δt12
Δt23
t1
t2
t3
(a): Biological perspective
(b): Simulation perspective
tA
tB
ΔtNN
tbio
N1
N2
N3
N4
ΔtN

Fig 6: Time models itself

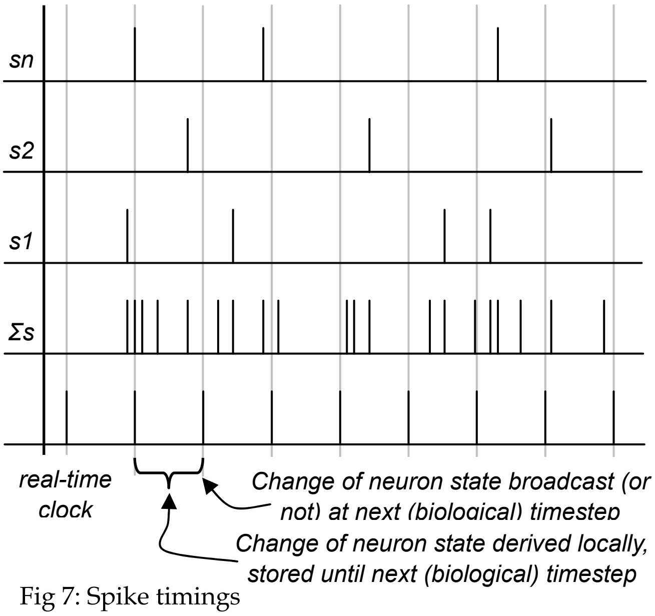
sn
s2
s1
Σs
real-time clock
Change of neuron state broadcast (or not) at next (biological) timestep
Change of neuron state derived locally, stored until next (biological) timestep

Fig 7: Spike timings

the synapse state local to neuron model B on node N4 represents Δt_{12} and Δt_{23} from fig 6a.

- The neuron-core mapping (A→N1, B→N4) is a function of the configuration software, and independent of biology.
- The node-node wallclock hop delay Δt_N (***O***(100ns)) is a function of the underlying hardware and the real-time traffic density - both decoupled from biology.
- The neuron-neuron wallclock delay Δt_{NN} (maximum ***O***(10us) - section 4) is a function of the configuration software, traffic density and SpiNNaker engine size - all decoupled from biology.

How might we bring all this together to capture the passage of *biological* time? From section 3, recall that there are different *types* of interrupts in the system. (The SpiNNaker architecture supports 32 distinct interrupt types, but only two are relevant here.)

- Each *core* is provided with a **packet handling interrupt** (invoked by an incoming packet representing a spike from an incident neuron).
- Each *node* asserts on each *core within it* a '**biological clock tick interrupt**'. These clocks (one for each node) are not phase locked, they are (computationally) slow (***O***(kHz)) and they indicate (to the incident cores) that biological time is passing. From a biological perspective, kHz is a reasonable speed, but from the perspective of the SpiNNaker cores, these 'biological time' events are rare - they occur around every 200000 machine cycles.

Returning to fig 6a, from a biological perspective:

- t_1 : A fires when it fires
- Δt_{12} : Pulse propagates to B
- t_2 : Pulse arrives when it arrives
- Δt_{23} : B integrates the incoming pulse(s)
- t_3 : B fires when (if) it fires

There is no synchronising clock; the sequence of actions is event-driven data-push.
From a simulation (SpiNNaker) perspective (fig 6b):

- t_A : A fires when it fires
- Δt_{NN} : Pulse (packet) propagates to B
- t_B : Pulse arrives O(us) later; triggers 'packet arrived' interrupt

In parallel with this (and not synchronised with it)

- t_{bio} : triggers a 'clock tick' interrupt with each tick

The meaningful interplay of wallclock (biological) and simulated time is achieved by the handlers below:

```
void OnPacket(MC_t *)
{
Remove packet from input buffer;
Store packet in synapse (age=0);
}
```

```
void OnTimer(TT_t)
{
Increment age of buffered packets;
If any 'arrived' (age == synapse delay),
then assert onto neuron state equations;
Integrate (one timestep) neuron state equations
Fire if necessary;
```

The **packet arrival handler** (which one might *prima facie* expect to do most of the work) in fact does nothing except remove the packet from the communication fabric and store it in a local buffer; the **clock tick handler** performs the necessary integration and subsequent pulse firing decisions, but this is done at a rate effectively throttled by the (biological) wallclock timer ticks. Referring to fig 7, the *individual* message frequencies ($S_1..S_n$) are much lower than that of the real time clock; the *superposition* of all these inputs (which may number in the thousands) have an exact timing which is a function of the neuron:core map, i.e. independent of biology (which is undesirable) BUT the message latency is much less than the biological time constants so it does not affect the biological fidelity of the simulation outcome (which compensates).

This all works because

- Biological wallclock time is modelled locally at each node (and thus each neuron model held within it) - the problem of system-wide global clock synchronisation is overcome (it is irrelevant).

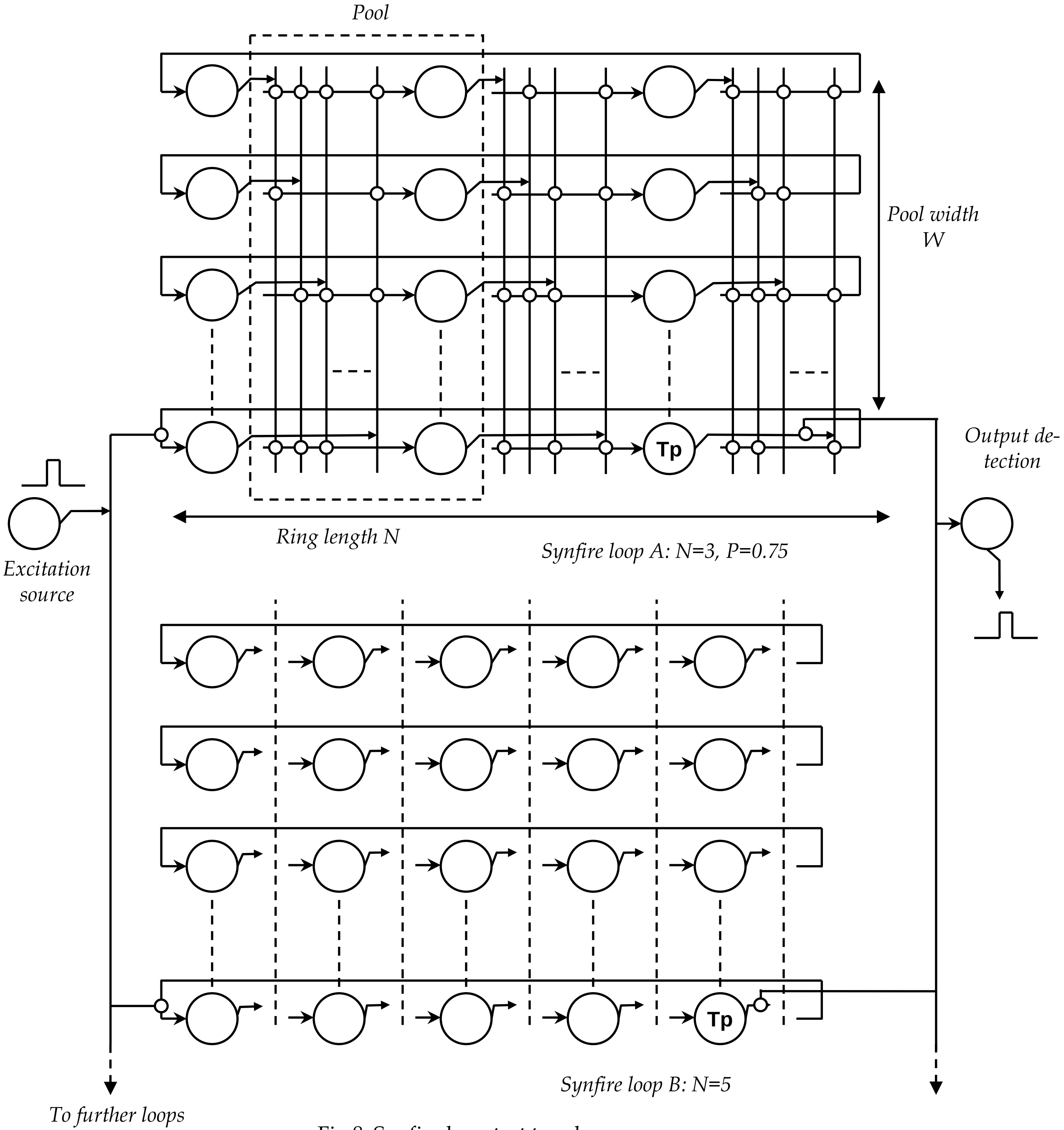

Fig 8: Synfire loop test topology

- At each wallclock time tick:
  - ○ Inputs are asserted (if age suitable)
  - ○ Equations integrated
  - ○ States updated
- Wallclock packet transit delay is *negligible* and *ignored*
- Biological delay is captured in the target synaptic model state
- The differential equations controlling the model are not stiff[3]:
  - ○ All the time constants are large compared to the biological clock tick
  - ○ Almost every numerical integration scheme is stable in these conditions [23].

5 Performance

The whole point of any simulator is to answer questions about a model of a system that are too expensive, difficult or dangerous to put to the real system. To be useful, two essential conditions must be met:

- The computational model of the system under simulation must accurately reflect the underlying reality, at least in the aspects of interest to the user. There is a tradeoff here: building a model that captures every aspect of reality is usually

[3] The LIH is not considered stiff, because the discontinuity when the neuron fires is not usually modelled with an ODE. However, under certain conditions [26] the Izhikevitch model [21,22] can be considered technically stiff.

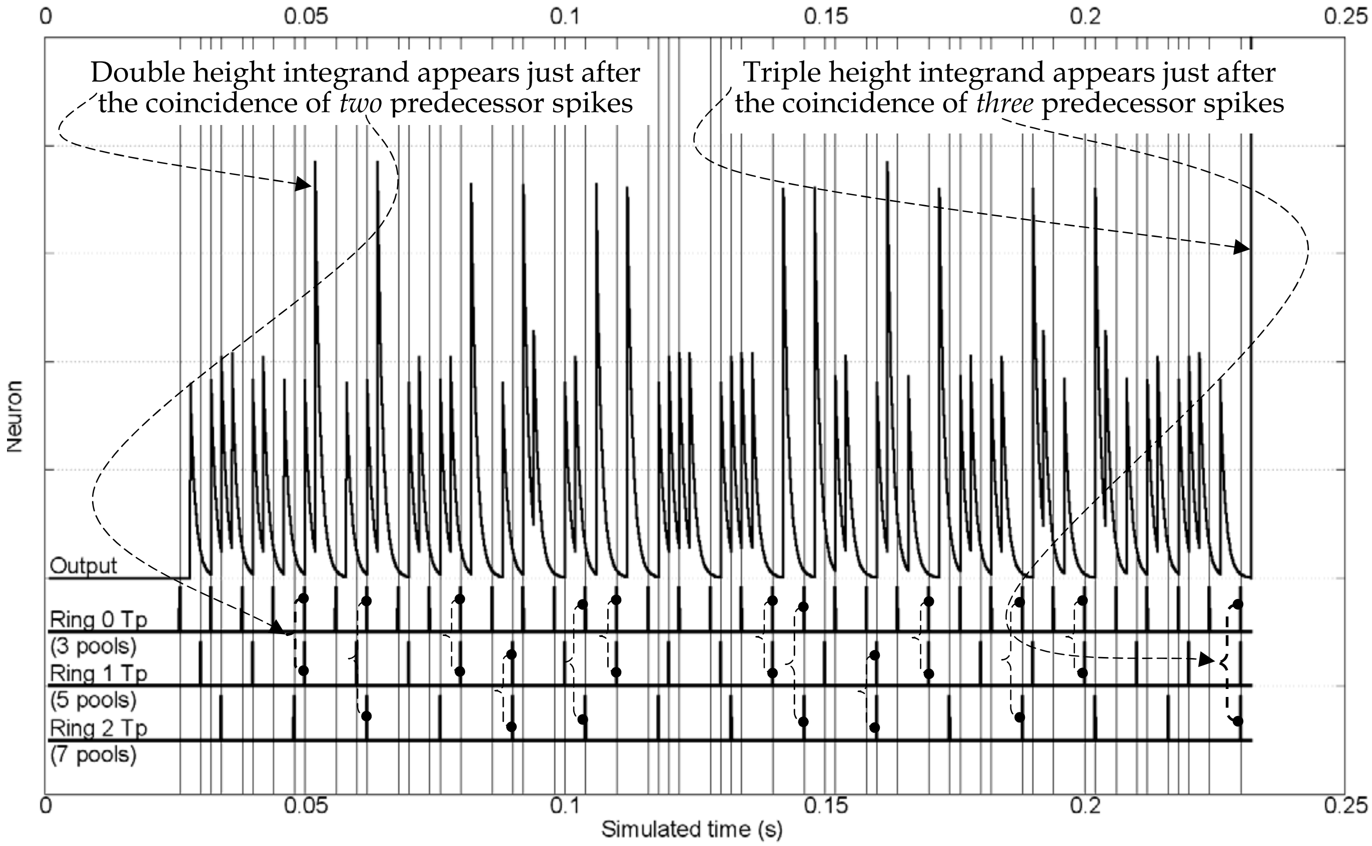

Fig 9: *Three* synfire rings beating (the experimental data of fig 10 uses *five*).

unnecessarily complicated and consequently places an unnecessarily high computation burden on the simulator; however, the ultimate purpose of any simulation is to answer questions to which the user does not know the answer, and it may be that some unexpected and superficially trivial component of a model has a dramatic effect upon the outcome (sensitivity, emergent behaviour).

- The simulation algorithms must correctly compute the *interaction* of the models used to build the system under simulation, and not introduce significant numerical artefacts of their own. (These effects may not be negligible on finite word length machines.)

SpiNNaker is a simulator. It is designed to simulate neural systems, and within that broad constraint, it does nothing that could not be achieved with a desktop machine. What it *is* intended to do (and what justifies the 140 person-years already invested in the development) is to perform meaningful (accurate) simulations on systems of hitherto unattainable size *in real time*. SpiNNaker is all about speed, or to be more precise, the scaling of compute cost with problem size. To provide a power perspective, SpiNNaker dissipates around 10nJ/synaptic event.

For a conventional simulation system, accuracy is assessed by simulating systems with known functionality ('reality' being obtained from physical measurement, the solution of closed-form mathematical representations, and/or comparison with other trusted simulation systems - preferably all three). Simulator performance (speed) is obtained by direct measurement.

SpiNNaker represents a significant departure from conventional simulation and computing system architectures, and so it is essential to explore the simulated behaviour of artificial circuits (those for which we understand functionality and hence the correct outputs) before deploying on systems for which we do not know the behaviour.

Neural systems are intrinsically noisy (section 3.3), and it is important to be able to distinguish between 'genuine' noise (i.e. a faithful simulation of an intrinsically noisy physical system) and artefacts introduced by the simulation technique; the former is useful, the latter destructively not so. The interpretation of non-trivial simulation results without noise is not easy (see, for example, the "non-firing" spike heights in fig 9). For this reason, we restrict ourselves here to a portfolio of noiseless, simple topologies whose correct behaviour is discernable by inspection. This gives us confidence that the output of the simulation is correct, even in the presence of obfuscating noise [19].

5.1 Simulator benchmarks

The base-level unit of our test set is the *synfire ring*[2,3] - see fig 8. This consists of a set of neural *pools* connected in a ring. Each pool consists of a set of neurons (W in size), the inputs of which are driven from some subset of the outputs of the preceding pool. A parameter P (the 'connection probability') indicates the fraction of outputs of each pool connected to the inputs of the subsequent pool.

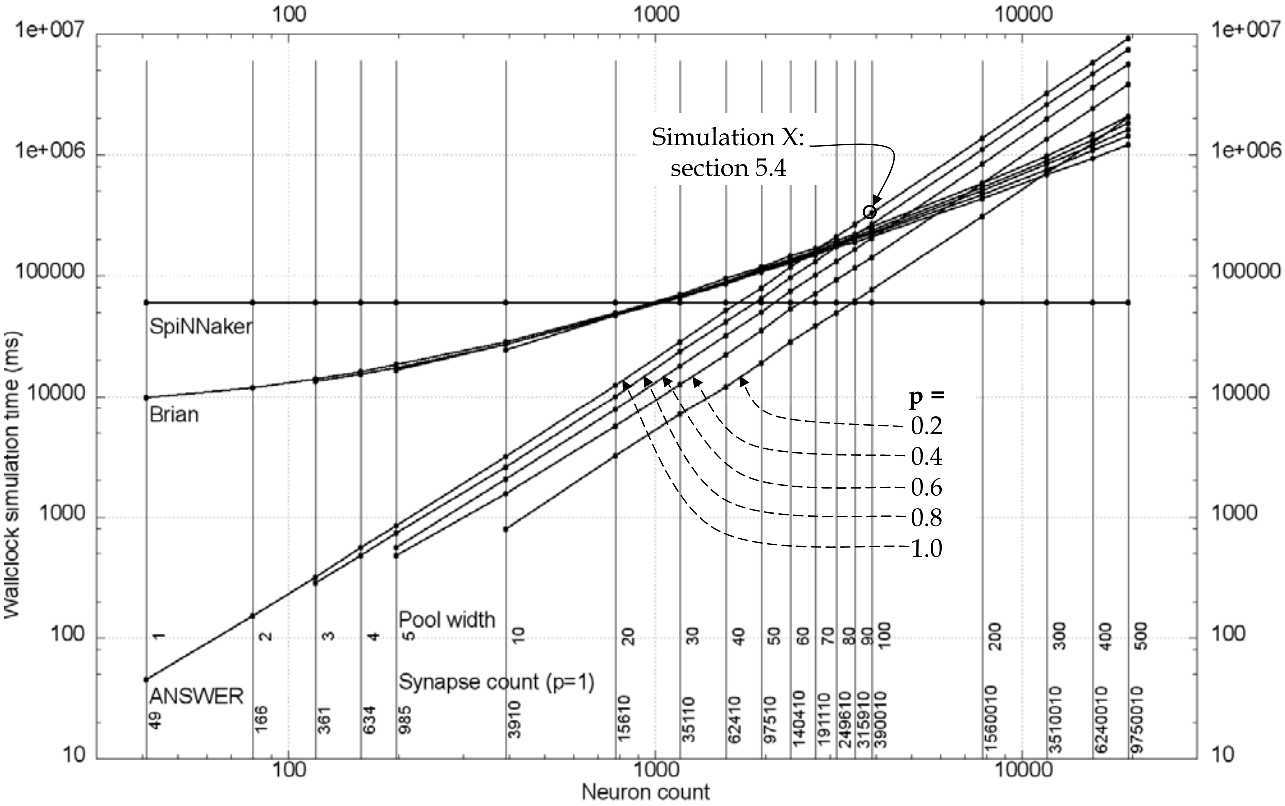

Fig 10: Wallclock simulation times

In the degenerate case (P=1) every neuron in a pool drives every neuron in the subsequent pool. The behaviour of such a ring is easy to understand: if no excitation is applied, nothing happens. However, if one neuron is triggered - for whatever reason - then thresholds and refractory periods allowing, a synchronised excitation wave will propagate round the ring indefinitely. The behaviour is tolerant of dropped packets (conveniently modelled by setting P≠1). If we take a *set* of rings - each of which has a different size (the sizes are relatively prime to prevent beating) - connect them in parallel (as in fig 8), and set the output device threshold such that it only fires when coincident pulses arrive from *all* rings simultaneously, we have a potentially formidable (from the point of view of the simulator) circuit, but one whose behaviour (from the analytic point of view) is entirely predictable.

The output "device" is a neuron model, the parameters of which are unimportant, except inasmuchas the threshold is defined as above, and the decay rate of the neuron state is sufficiently large to accommodate slight offsets in the arrivals of the Tp signals.

The I/O delay and neuron count for the test circuits is shown in Table 1. The generalized expressions are

Ring set period = $\Pi_i\, prime_i$
Ring set neuron count = $W * \Sigma_i\, prime_i$

Choosing the sizes of the rings in each set to be relatively prime (and making them absolutely prime is a simple way of achieving this) means that the time of the final output spike (corresponding to the first coincidence of each individual ring output) is simple to predict.

Table 1
Synfire ring sets

Set	Ring size	Input / output delay	Neurons
1	{3}	3	3W
2	{3,5}	15	8W
3	{3,5,7}	105	15W
4	{3,5,7,11}	1155	26W
5	{3,5,7,11,13}	15015	39W
6	{3,5,7,11,13,17}	255255	56W
7	{3,5,7,11,13,17,19}	4849845	75W

5.2 Quantitative temporal behaviour

Fig 9 shows the device activity associated with the first device in each pool for a three ring {3,5,7} system. The bottom three traces depict the outputs from a single neuron in each of the final (right-hand) pools in each ring. Exactly which neuron is chosen is largely irrelevant (assuming W and P are not pathologically low) and without loss of generality we choose the neurons in each pool in fig 8 labelled as 'Tp' (**T**est **p**oint). With a neural delay of 2ms, the *period* of ring 0 is shown as 6ms, that of ring 1 is 10ms and that of ring 2 as 14ms, giving the first coincidence at *fire_delay* x (3x5x7) = 210ms after the initial trigger at t=20ms. The fourth trace in fig 9 ('output') shows the internal integrand (state) of the output neuron in fig 8. The threshold of this neuron (and this neuron alone) is set to the system spike height x ring count, turning it into a simple coincidence detector. This neuron fires only when

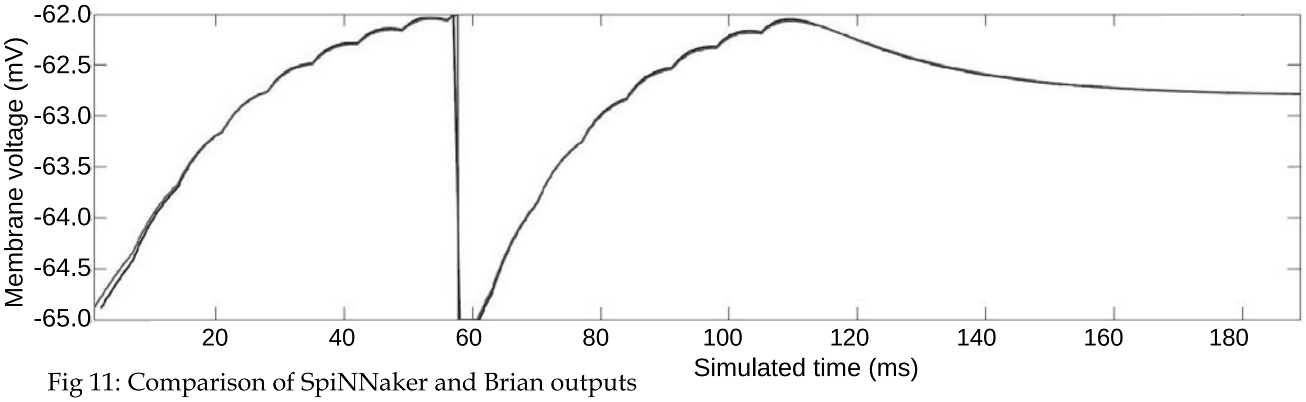

Fig 11: Comparison of SpiNNaker and Brian outputs

output spikes from all the rings in the test configuration fire in unison (thereby providing the timing signal that ends the experiment).

Fig 10 shows the wallclock times for the simulation of ring set 5 (see Table 1) for three simulators. Three sets of wallclock timing curves are presented, each for a different value of pool interconnection probability P. Synaptic counts corresponding to P=1 are shown as numeric labels at the appropriate abscissa positions; the synapse counts for any of the curves P≠1 may be obtained by multiplying the P=1 value by P.

- SpiNNaker: Here all the P curves lie exactly on top of each other - time models itself, so we would expect the responses to be independent of the synapse count, which is the case. (The full SpiNNaker machine is currently being assembled; these results are taken from a 768 core subsystem.) The wallclock time is equal to the simulated time, and independent of the pool width, as expected.

- BRIAN: These are the wallclock times for BRIAN[1], a widely used neural simulator running on a conventional architecture (here a single-thread implementation running on an Intel i7 2.3GHz machine). Here, the wallclock times grow with the problem size. For small circuits, BRIAN operates faster than biological (real) time, and the absolute speedup will obviously depend on the host computer. The reason that the BRIAN curves grow slowly and are not highly dependent on the overall synapse count is that internally the system represents the interconnection topology as a matrix, and uses an extremely fast and highly tuned matrix multiply class to propagate data within it. The performance is only weakly dependent upon the proportion of non-null entries in the connection matrix.

- ANSWER: uses an extremely unsophisticated LIF model and a generic event pump, so the broad characteristics of the behaviour - extremely fast for small systems, growing faster than BRIAN as the problem gets larger - are unsurprising. ANSWER yields to SpiNNaker when the overall problem size exceeds around 60000 synapses.

Note that the curves for P≠1 do not extend all the way to the smallest synapse count. This is because the statistical nature of the pool-pool interconnect regime has open-circuited some of the rings, meaning that the simulation never stops. As P→0, this becomes more of an effect with low pool widths.

The data in fig 10 is provided to put the performance of SpiNNaker into context. The ANSWER curves represent the edge of the performance envelope obtainable by an event-based single thread architecture. The horizontal line (the superposition of *all* SpiNNaker data) represents a physical implementation of the Holy Grail of parallelisation: Amdahl's law notwithstanding, the realised behaviour of the system scales as a constant - 1:1 with wallclock time (up to some finite limit). It is an elegant demonstration of the Gustafson-Barsis principle.

5.3 Correctness

Does the output of SpiNNaker, fast though it may be, actually agree with reality and other established simulator results? As an illustration of the correctness of the approach embodied in SpiNNaker, fig 11 shows the internal state of a neuron at an arbitrary point within a simulation. *What* the neuron is doing is unimportant; what is relevant is the close agreement between the values for the internal neuron state as computed by BRIAN and SpiNNaker. The very small discrepancy at t ≈ 58ms is a reporting artefact (SpiNNaker is a *real-time* machine, and within the domain of neural simulation, the time taken to exfiltrate data has a small quantitative effect on the recorded outputs).

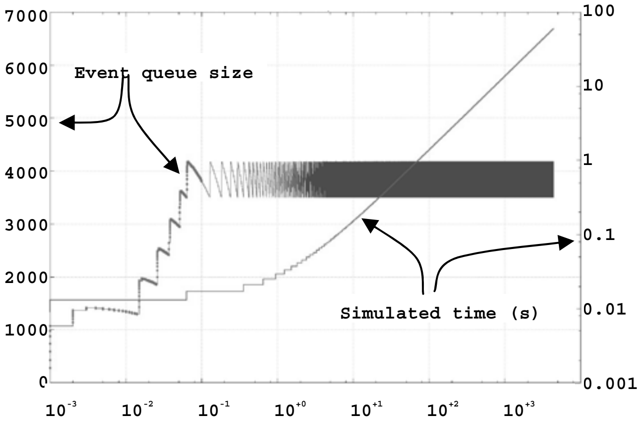

Fig 12: Dynamic memory footprint of a single-thread simulation of synfire ring circuits (fig 8).

5.4 Resource utilisation

One of the many consequences of shrinking machine geometries is a change in the *distribution* of compute costs: it now takes orders of magnitude more time and energy to move data about than it does to process it[14]. This is one of the drivers for many-core architectures: problem data (state) may be physically distributed across a machine, and processed - hopefully/usually - by a core physically adjacent to it. The full SpiNNaker engine contains ~ 7TByte of memory, *distributed throughout the physical machine*, and embedded within it are the million+ cores. The design is realized from relatively low-performance devices, and yet achieves a memory access bandwidth of 28.8 Gbyte/s per node, an impressive figure even by the standards of today. BRIAN and ANSWER are conventional software systems running on single-thread fetch-execute architectures.

The data footprint (in all three systems) consists of two components: the state data of the system-under-simulation and the overhead structures necessary to organize and permit efficient access to this data. This overhead is lower for SpiNNaker because the state is physically distributed over the computer mesh (fig 1) and at that level of granularity does not require organizing, because the individual node memories are independent and access to them not coherent.

Fig 12 shows the dynamic memory footprint for the specific simulation X in fig 10 on ANSWER, as a function of simulation time - note the logarithmic abscissa. (The *total* footprint at any (wallclock) time is the sum of this figure and the storage overhead.)

The exact nature of the data in fig 12 is completely dictated by the connectivity of the circuit under simulation (fig 8) and is, at that level, uninteresting. The point - in the context of this paper - is that the dynamic memory data in fig 12 shows - at any wallclock time - the quantity of 'future (simulated) events' to be processed by the simulator[4].

By virtue of the massive hardware parallelism afforded by the SpiNNaker architecture, the need for this memory simply does not arise - SpiNNaker handles all events calculated to occur at simulated time ***t*** at wallclock time ***t***.

5.5 Limits of performance

Within the region of intended usage for SpiNNaker, simulation time scales 1:1 with wallclock time; this is impressive, but clearly not sustainable for indefinitely large networks on limited hardware. The design intent is that maximum density of representation is 1000 neuron models hosted per core. Within each core, the interplay of spike and clock interrupts is described in section 4. Although the biological clock interrupts occur relatively rarely to the processor clock (around one biological tick to 200000 processor clock ticks), within this interval, the processing of interrupts is inevitably serialised within the core. In the worst case, then, (if every neuron hosted by a core is simultaneously excited), an individual core has 200000/1000 = 200 processor cycles to handle all necessary computation. Thus the natural end point to the SpiNNaker lines in fig 10 occurs when neuronal representation becomes so dense that the core simply runs out of time to process all the spike interrupts between biological clock tick events. SpiNNaker is a massively parallel machine - when completely assembled, it will consist of over a million cores - and each of these is independent and parallel in execution. The speed performance depicted in fig 10 will not change when the larger machine is assembled - simply the capacity of the simulator will grow.

This pathology can be ameliorated to a certain extent by slowing the biological clock, but this strategy starts to impinge on the validity of the numerical assumptions of section 4, and can only be taken so far.

A second design limit that can cause problems is that of *packet dropping*. The cores of the SpiNNaker engine are truly independent and parallel; there is no overseer or meta-network to impose traffic control. Once a packet is launched, it is swept up by the hardware communications infrastructure and delivered to its destination. The precise hop sequence of each packet trajectory is determined by the configuration software before the simulation starts. The asynchronous nature of the hardware supports considerable in-flight packet buffering (distributed along the route), but it is still possible for traffic density transients to momentarily overwhelm a router. When this happens, the packet is removed from the communications fabric, and placed in a register *local* to the router that dropped the packet. At the same time, an interrupt is raised to the monitor core of the node containing the router/dropped packet; this handler can remove the packet to the software stack, and re-inject it into the communications structure at a later point when the local packet traffic density subsides. SpiNNaker is designed with a safety margin of a factor of ten - i.e. the hardware can handle packet rates around ten times the expected average density, but it has been found possible to (transiently) exceed this figure. Re-injecting dropped packets in the manner described ensures their eventual arrival, although not necessarily within the correct biological timeslot. A natural limit to this extenuation occurs due to the finite size of any local packet buffer. Utilizing one of the 16 application cores (and the associated DTCM) as a dedicated 'recover and re-inject' thread allows a buffer size of ~ 40k packets/node.[5]

In the end, the real-time simulation capabilities of SpiNNaker can be overwhelmed by excessively dense representation of neural circuits or excesses of local communications traffic - but so can any simulation engine. SpiNNaker contributes to the art by extending the boundaries of real-time simulation to hitherto inaccessible scales.

6 Final comments

These initial quantitative assessments of the comparative performance of SpiNNaker show that the machine dem-

[4] ANSWER is totally internally instrumentable, but the corresponding information relevant to BRIAN is much coarser grained and adds nothing to the discussion.

[5] The SpiNNaker dies are fabricated with 18 ARM cores: on power-up, a self test locates 16 functioning cores. If there are more (and experience shows that >95% of dies have 17 or 18 fully-functional cores), then no application resources need be sacrificed for this, and we obtain a flat free performance boost.

onstrates the (weak) scaling that was one of its original design criteria - the simulation time scales 1:1 with wallclock. This scaling maintains real-time performance independently of the size of the problem by increasing the deployed hardware resource in proportion to the computation load; this is sustainable due to the architectural innovations, (inspired by computation mechanisms in the brain), that allow the computation to work with a non-deterministic communication infrastructure, globally asynchronous operation (time models itself), and no memory coherency mechanism.

SpiNNaker has been designed specifically for modelling large-scale systems of spiking neurons in biological real time, leading to the architectural features described here. Architectures of this type are suitable for wider applications of similar nature, though inevitably these must be extensively restructured to map them onto the machine.

SpiNNaker has absorbed considerable research effort. Nevertheless, these are dwarfed by the resources poured into the BlueBrain project[15], the Human Brain project[15], BrainScaleS[4] and others. It does nothing functional that cannot be achieved (slowly) on a 3kGBP desktop machine (albeit one containing 7Tbytes of memory), but it delivers results many orders of magnitude faster than the desktop - in real time. It is a research machine, and - like any simulation system - can be overwhelmed by problem sets that take it out of its intended region of operation. However, within that arena, its capabilities provide computational neuroscience with a resource unlike any other currently - or foreseeably - available.

The machine does not represent a panacea even just for neural modelling, as it has its limitations. Most notable among these is the tendency of the discrete time-step software implementations to cause communication traffic to be bursty so that, even if the average traffic is well below capacity, transient bursts at the start of each millisecond interval can overload the fabric, resulting in (recoverable) dropped packets. In networks with particularly pathological transient traffic characteristics the user may have to take measures to reduce the intensity of network traffic peaks.

Overall, SpiNNaker represents an unprecedented capability to simulate large-scale spiking neural networks running in biological real time, and as such it achieves two significant goals: 1) it moves forward the computational barriers to detailed brain network modelling by a significant distance, and 2) it opens the door to, and demonstrates the validity of, event based computation as a valuable technique (almost) orthogonal to conventional computing engines: non-deterministic message passing, non-coherent memory access and globally unsynchronised hardware.

Acknowledgment

The design and construction of the SpiNNaker machine was supported by EPSRC (the UK Engineering and Physical Sciences Research Council) under grants EP/D07908X/1 and EP/G015740/1, in collaboration with the universities of Southampton, Cambridge and Sheffield and with industry partners ARM, Silistix and Thales. Ongoing development of the software is supported by the EU ICT Flagship Human Brain Project (FP7-604102), in collaboration with many university and industry partners. Our own exploration of the capabilities of the machine is supported by the European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 320689.

References

[1] D. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in Python" *Frontiers of Neuroinformatics* 2:5 doi: 10.3389/neuro.11.005.2008

[2] P. Zheng and J. Triesch, "Robust development of synfire chains from multiple plasticity mechanisms" *Frontiers of computational neuroscience*, 2014 doi: 10.3389/fncom.2014.00066

[3] G.L. Gerstein, E.R. Williams, M. Diesmann, S. Grun and C. Trengrove, "Detecting synfire chains in parallel spike data", *J. Neurosci Meth,* vol 206, no 1, pp 54-64 doi: 10.1016/j.jneumeth.2012.02.003

[4] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grübl, J. Schemmel, R. Schüffny (2007). "Wafer-scale VLSI implementations of pulse coupled neural networks." *Procs IEEE Conf on Sensors, Circuits and Instrumentation Systems.*

[5] Brett & Gerstner (2005) "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity". *J Neurophysiology* 94:3637-3642.

[6] S. Scholze, S. Schiefer; J. Partzsch; S. Hartmann, C. Mayr, G. Höppner, H. Eisenreich, S. Henker, B. Vogginger, R. Schüffny "VLSI implementation of a 2.8 Gevent/s packet-based AER interface with routing and event sorting functionality" *Frontiers in neuroscience*, 2011, 5

[7] A. Rast, J. Partzsch, C. Mayr, J. Schemmel, S. Hartmann, L.A. Plana, S. Temple, D. Lester, R. Schüffny, and S. Furber. "A location-independent direct link neuromorphic interface" *Procs International Joint Conference on Neural Networks*, 2013

[8] U. Mallik, R.J. Vogelstein, E. Culurciello, R. Etienne-Cummings, G. Cauwenberghs "A real-time spike-domain sensory information processing system." *Circuits and Systems, IEEE International Symposium on*, 2005, 1919-1922

[9] R.J. Vogelstein, U. Mallik, E. Culurciello, G. Cauwenberghs, R. Etienne-Cummings, "A multichip neuromorphic system for spike-based visual information processing." *Neural computation*, 2007, 19, 2281-2300

[10] J. Park, T. Yu, C. Maier, S. Joshi, G. Cauwenberghs, "Hierarchical Address-Event Routing architecture for reconfigurable large scale neuromorphic systems." *Circuits and Systems (ISCAS), IEEE International Symposium on*, 2012, 708-711

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

[12] P.A. Merolla, J.V. Arthur, R. Alvarez-Icaza "A million spiking-neuron integrated circuit with a scalable communication network and interface", *Science*, 2014, 345, (6197), pp. 668-673

[13] R. Silver, K. Boahen, S. Grillner "Neurotech for neuroscience: unifying concepts, organizing principles, and emerging tools", *J. Neurosci.*, 2007, 27, (44), pp. 11807-11819

[14] P.J. Fox, S.W. Moore, S.J.T Marsh "BluehiveVA field-programmable custom computing machine for extreme-scale real-time neural network simulation" *Proc. IEEE 20th Int. Symp. Field-Programmable Custom Comput.*, March 2012, pp. 133-140

[15] H. Markram et al, "Reconstruction and simulation of neocortical microcircuitry" *Cell* vol 163, 2015, pp 456-492.

[16] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown and S. B. Furber, "SpiNNaker: A 1W 18-core System-on-Chip for Massively-Parallel Neural Network Simulation", *IEEE Journal of Solid State Circuits*, (48)8, pp 1943-1953 Aug 2013

[17] S. B. Furber, D. R. Lester, L. A. Plana, J. Garside, E. Painkras, S. Temple, A. D. Brown, "Overview of the SpiNNaker System Architecture" *IEEE Transactions on Computers*, (62)12, pp.2454-2467, doi: 10.1109/TC.2012.142

[18] A.D. Brown, S.B. Furber, J.S. Reeve, J.D. Garside, K.J. Dugan, L.A. Plana, and S. Temple, "SpiNNaker - programming model" *IEEE Transactions on Computers*, (64)6, pp 1769-1782, doi: 10.1109/TC.2014.2329686

[19] A.D. Brown, R. Mills, K.J. Dugan, J.S. Reeve and S.B. Furber "Reliable computation with unreliable computers", *IEE Computers and Digital Techniques*, 2015, (9)4, pp 230-236, doi: 10.1049/iet-cdt.2014.0110

[20] AMBA Design Kit Technical Reference Manual, ARM DDI 0243A, 2003.

[21] E.M. Izhikevich, "Simulation of Large-Scale Brain Models", 2005. *www.nsi.edu/users/izhikevich/interest/index.htm*

[22] E.M. Izhikevich and G.M. Edelman, "Large-Scale Model of Mammalian Thalamocortical Systems" *Proc. National Academy of Sciences of the USA*, vol. 105, no. 9, pp. 3593-3598, Feb 2008, doi: 10.1073/pnas.0712231105.

[23] E. Kreyzig, *Advanced Engineering Mathematics 9th edition*, J. Wiley, ISBN-13: 978-0-471-72897-9

[24] W. Humphrey, A. Dalke and K. Schulten, "VMD - Visual Molecular Dynamics" *J. Molec. Graphics* (14)1 pp 33-38 1996.

[25] A.N. Burkett, "A review of the integrate-and-fire neuron model" *Biological Cybernetics*, (95)1 pp 1-19 2006

[26] M. Hopkins and S.B. Furber, "Accuracy and efficiency in fixed point neural ODE solvers", *Neural computation*, 2015, (27)10.

[27] H. Markram et al, "The Human Brain Project", April 2012

[28] D. Attwell and S.B. Laughlin, "An energy budget for signalling in the grey matter if the brain", *J. Cereb. Blood Flow Metabolism* 21(10) Oct 2001 pp 1133-1145.

[29] D. Bray, "Cell Movements", *Garland publishing* 1992 ISBN13 9780815307174

[30] Engineering and Physical Sciences Research Council, UK, grant ref EP/N031768/1.

[31] P. Bonzon "Towards neuro-inspired symbolic models of cognition: linking neural dynamics to behaviors through asynchronous communications" *Cogn Neurodyn* doi:10.1007/s11571-017-9435-3.

[32] S. Zeki "A massively asynchronous, parallel brain" *Phil. Tran. R. Soc. B*, 370:20140174.

[33] S.B. Furber "Large-scale neuromorphic computing systems" *J Neural Engineering* July 16 doi: 10.1088/1741-2560/13/5/051001

[34] P.E. Latham, A. Roth, M. Hausser and M. London "Requiem for the spike" *Soc. Neuroscience Abstracts*, vol 32:432.12 2006

II. Biographies

Andrew Brown (M’90, SM’96) is Professor of Electronics at Southampton University, UK. He has held visiting posts at IBM Hursley Park (UK), Siemens NeuPerlach (Germany), Multiple Access Communications (UK), LME Design Automation (UK), Trondheim University (Norway), Cambridge University (UK) and EPFL (Switzerland). He is a Fellow of the IET and BCS, a Chartered Engineer, and a European Engineer. Corresponding author.

John Chad is a Professor of Neuroscience at Southampton University. He has worked at UCLA, Los Angeles and the Sandoz Institute for Medical Research, USA His research interests include the control of neuronal excitability, neurodegeneration, neuronal self-organisation and neurocomputation. He is member of the British Neuroscience Association and the Physiological Society.

Raihaan Kamarudin is a PhD student in Electronics and Computer Science at the University of Southampton, UK. His PhD is on the the Development Trajectory of the *C.Elegans* neural system. He has BE and ME degrees from Takushoku University, Japan, and has lectured at the University Technical Malaysia.

Kier Dugan is a PhD student in Electronics at the University of Southampton, UK. His interests include self-configuration of distributed computing systems, high-performance networks and computer architecture. He has an MEng in Electronic Engineering from the University of Southampton.

Steve Furber (M'98, SM'02, F’05) is ICL Professor of Computer Engineering in the School of Computer Science at the University of Manchester. He worked at Acorn Computers where he led the development of the first ARM microprocessors. He is a Fellow of the Royal Society, the Royal Academy of Engineering, the BCS and the IET.