Correctness and Performance of the SpiNNaker Architecture

Thomas Sharp and Steve Furber

Abstract—SpiNNaker is a computer architecture designed to simulate many millions of neurons in real-time, using very many low-power processors and biologically inspired communications. This paper demonstrates that prototype SpiNNaker hardware correctly and quickly simulates networks of point neurons with respect to established reference simulations. Models of increasing complexity are presented and the simulation results obtained from SpiNNaker, NEST and Brian are shown to correlate. An execution profile is sketched of real-time simulation on SpiNNaker, and it is shown to outperform NEST using similar computational resources on a standard benchmark model.

I. INTRODUCTION

SIMULATION of neural tissue is a promising methodology for investigating the function of the brain. However, the feasible scale and speed of simulations is currently limited by enormous parallelism and power disparities between biological and silicon computers. SpiNNaker is designed to advance these limits, using up to a million processors and spike-based communication hardware to simulate up to a billion neurons and a trillion synapses in real-time.

The SpiNNaker chip contains eighteen processors, each to simulate between one hundred and one thousand neurons, and a packet-switched router that conveys simulated action potentials between local processors and those on distal chips [7]. Each processor runs an application run-time kernel that may be configured to simulate arbitrary neuron and synapse dynamics using an associated application programming interface [10]. Simulation takes place in real-time and is driven by timer events that prompt neuron-state computations and packet events that prompt synaptic-current computations. All computations are performed in fixed-point arithmetic, and synaptic weights are stored in an off-chip memory.

This paper demonstrates that this approach affords correct and rapid simulation of neural tissue with respect to established reference simulators. Experiments are presented that compare the output and performance of SpiNNaker, NEST (v2.1) and Brian (v1.3.1) using models specified in the PyNN (v0.7) simulator-abstraction language. The results show that SpiNNaker correctly simulates each model and does so more quickly than NEST given similar computational resources. A profile of the code running on each processor explains the achievable real-time throughput of the architecture.

II. SIMULATION ACCURACY

A series of experiments were performed on SpiNNaker to test the function of the API and to verify the accuracy of the leaky integrate-and-fire (LIF) model implementation.

Experiments of increasing complexity were repeated on the SpiNNaker, Brian and NEST simulators and the results were compared. Except where otherwise indicated, SpiNNaker is represented by blue traces, NEST by red, Brian by green, and any analytical solution by cyan. The LIF model was consistently instantiated with the parameters: $E_L = -70\,\text{mV}$, $V_{\text{reset}} = -70\,\text{mV}$, $V_{\Theta} = -50\,\text{mV}$, $\tau_m = 40\,\text{ms}$, $R_m = 50\,\Omega$, refractory period $= 1\,\text{ms}$, $\tau_{s_e} = 20\,\text{ms}$, $\tau_{s_i} = 5\,\text{ms}$.

A. Response to rheobase current

Rheobase current is defined as the minimum constant input required to elicit a spike in a neuron. The rheobase of a biological neuron is found with a sharp electrode by applying some current for a few hundred milliseconds, checking for a spike, and repeating as necessary. The abstract nature of the LIF neuron allows for a more rigorous definition, by evaluating the analytical form as $t$ tends to infinity

$$
\lim_{t \to \infty} V(t) = E_L + R_m I - R_m I e^{-t/\tau_m}
$$

and simply setting $V(t) = V_{\Theta}$ and solving for $I$

$$
I = \frac{V_{\Theta} - E_L}{R_m}
$$

To elicit a spike in finite time, a small quantity must be added to $I$; in this experiment, $I$ was incremented by one picoamp.

A single neuron was simulated for ten seconds under rheobase current to compare error in membrane potential computations, in terms of spike times, between the three simulators and an analytical solution described by Dayan and Abbott [3]. Figure 1 shows the membrane potential traces from each simulator around the first spike (top) and traces from each simulator around the first spike (top) and the cumulative error in spike times with respect to the
analytical solution (bottom). It is apparent that NEST adheres to the analytically derived spike times, while a small drift in the computations performed by SpiNNaker and Brian cause firing two and three milliseconds early respectively. This drift is curtailed by the occurrence of spike, which resets the membrane potential to the common value of \( V_{\text{reset}} \), so the timing error is discretised and successive spike times diverge linearly. In the absence of such spikes, membrane potential computations may drift more significantly; the subsequent experiment sought to test this hypothesis.

**B. Response to postsynaptic currents**

To evaluate computation drift over longer intervals, and to test the computation of postsynaptic currents, a single neuron was simulated under input spikes from one excitatory and one inhibitory presynaptic neuron. The interspike intervals and synaptic weights of the presynaptic neurons were tuned to ensure that postsynaptic membrane potential remained sub-threshold throughout the simulation. The output of the three simulators was compared to an analytical solution presented by Brette et al. [1] that, unlike the solution given by Dayan and Abbott, accounts for time-varying postsynaptic currents. The absence of output spike times for comparison required that membrane potentials were compared directly using the Pearson correlation coefficient over a sliding window of width equal to quarter of the input interspike interval.

Figure 2 shows that the membrane potential traces from all three simulators closely match the analytical solution even after ten seconds of simulation. Contrary to the hypothesis, the correlation between computed membrane potentials is strong and periodic, with a frequency equal to that of the combined input spike trains. Furthermore, the computations are robust to perturbation: figure 3 shows that particular patterns of input spikes may be used to elicit an erroneous output spike on certain simulators, but membrane potential traces quickly reconverge regardless.

Results from different simulators converge in this experiment because the state variables are exponentially stable and output spikes do not affect the timing of subsequent input spikes. However, in networks with recurrent projections, output spikes do contribute to subsequent inputs, which suggests that simulations of such networks on different platforms may produce greatly differing results.

**C. Response to recurrent projections**

To examine the effect of recurrent projections on spike times, the simple network of excitatory and inhibitory neurons shown in figure 4 was simulated. Excitatory neurons received rheobase current input, and projection probabilities and synaptic weights were chosen solely to cause simple oscillatory activity in the network. Although the projections were specified probabilistically, the synapses comprising each projection were chosen deterministically to ensure that an identical network was deployed across all simulators. Blue, red and green denote the firing of excitatory, inhibitory and stimulus cells in simulation traces; labels disambiguate between simulators where necessary.

Figure 5 shows the spike times and firing rates obtained from simulation on each platform. As expected, the results produced by each simulator differ; spikes may be easily identified that occur in one simulator and not in either of the others. Indeed, after the first five hundred milliseconds it is practically impossible to identify any spike that is common to all three simulators because the dynamics of recurrently connected networks are chaotic and, consequently, a quantitative comparison of spike times from different simulators...
is futile [4]. Qualitatively, some correlation can be observed in the firing rates of the three simulators in the first 150 milliseconds of activity, although SpiNNaker then deviates from the other two.

Networks are rarely simulated under such unstructured input. To examine firing rate correlations under more likely input, structured stimulation was delivered to the network by an artificial spike source. A set of eighty spike trains was targeted upon the excitatory population, with a projection probability of 0.1 and synaptic weights of 0.1 nanoamps. The firing times of the spike trains was determined by a Poisson process, in which firing rate varied sinusoidally with a period of 200 milliseconds and an amplitude of 20 hertz. The PyNN interfaces to both Brian and NEST contained bugs that made such simulations impossible; for the purposes of this experiment, the PyNN-NEST interface was fixed and Brian was disregarded.

Figures 6 and 7 show key comparisons [1] of the simulation results obtained from SpiNNaker and NEST. The former shows the Pearson’s correlation coefficient for both the excitatory and inhibitory firing rates, calculated with a sliding window of 25 milliseconds. The latter shows the histogram of excitatory interspike intervals, taken with bin sizes of approximately five milliseconds. Both metrics show strong similarities in the outputs of the two simulators: firing rates remain strongly correlated, with only periodic divergences, after five full seconds of simulation; the interspike intervals clearly follow a similar bimodal distribution.

Brunel [2] simulates a recurrently connected, externally driven network of leaky integrate-and-fire neurons with delta-current synapses in order to understand the fundamental
properties of such models. He finds that the firing rate depends primarily on the ratio of excitatory to inhibitory synaptic currents in the network, which is a function of the number of synapses of each type and their weight. In the network discussed, excitatory synapses outnumber inhibitory synapses four to one, so synaptic currents balance when the inhibitory-to-excitatory weight ratio $g$ is four. When $g$ is less than four, the network is hyperactive; when $g$ is greater than four, the network is hypoactive.

Simulations were conducted to reproduce the findings of Brunel in NEST and SpiNNaker, using leaky integrate-and-fire neurons with exponential-current synapses. A model was constructed of eight hundred excitatory and two hundred inhibitory cells, in which every cell received a synapse from every other with a probability of 0.1 in addition to synapses from one hundred external drivers spiking at 25 hertz. Refractory periods were set to ten milliseconds to limit maximum spiking frequency. Excitatory weights were fixed at 0.1 nanoamps and inhibitory weights were varied a value of $g$ ranging from zero to eight in successive trials. The average firing rate across all excitatory cells was recorded.

Figure 8 shows the firing rates achieved by SpiNNaker and NEST as a function of the weight coefficient $g$. The simulators produce results in agreement with both Brunel and one another, in that firing rate varies sigmoidally as a function of $g$ from near-maximum to near-minimum.

D. Discussion

These experiments show that SpiNNaker simulations are faithful to results obtained from established simulators. Under constant current input current, SpiNNaker evaluates neuron equations almost as accurately as NEST and more so than Brian. The same may be said for periodic exponential synaptic currents, where there is no long-term divergence in results from each simulator. Under asynchronous nonrecurrent synaptic inputs, SpiNNaker may emit a spike and reset membrane potential where NEST does not, but the exponentially stable neuron equations soon reconverge. Recurrent synaptic connections, however, amplify any differences in membrane potential evaluation, so spike-by-spike comparison of results across simulators becomes impossible [4]. This also manifests in higher-order observations of activity, in that there is little correlation between the firing rates of recurrently connected networks under unstructured input simulated on SpiNNaker, NEST and Brian. However, under structured input, networks simulated on SpiNNaker and NEST show strongly correlated firing rates and qualitatively similar distributions of interspike intervals. In parameter-sweeping experiments, these simulators agree with both one another and the literature that network firing rate is a function of the ratio between inhibitory and excitatory synaptic current amplitudes. Ultimately then, taking NEST as a reference point, SpiNNaker appears to be a reliable and correct simulator of recurrently connected networks.

III. PERFORMANCE PROFILE

A series of experiments were conducted to sketch a performance profile of SpiNNaker hardware and software. Each SpiNNaker processor has a core thread for executing software and a DMA thread for transferring data between memories. Simulations of neural tissue comprise neuron state computations that are performed exclusively by the core thread and synapse state computations that are performed by a pipeline through the core and DMA threads. As described in detail by Sharp et al. [10] the three-stage packet-processing pipeline begins with receipt by the core of a spike packet from the router, proceeds with a DMA transfer of corresponding synaptic weights to processor-local data memory, and concludes with translation by the core of synaptic weights into input currents to neurons. It is apparent a priori that this pipeline dominates core time: computing each neuron state costs approximately $2^7$ instructions, responding to each synaptic event, defined as one spike innervating one synapse, requires $2^5$ instructions; each processor is expected to handle $2^7$ neuron state computations and $2^{13}$ synaptic events per millisecond, so the latter computation clearly dominates. The most important performance profile for SpiNNaker, therefore, is that of the packet-processing pipeline.

A simple PyNN network was constructed in which a source population on one processor projected to target populations on one or more processors of the same chip. In each trial, the source population was configured to emit some number of spikes in every millisecond, which innervated some number of synapses on each of the target populations. At the start of each millisecond, each target processor counted and flushed the excess packets that were left unprocessed from the previous millisecond; these data were reported to the host at the end of the simulation. From the parameters and excess counts of each trial, it was possible to infer the pipeline throughput.

Jin et al. [5] set a per-processor performance target of 1024 optimised Izhikevich neurons with delta-current synapses and ten thousand synaptic events per millisecond. The leaky integrate-and-fire neurons with exponential-current synapses used here are fourfold as costly, so only 256 cells are simulated per processor. Successive trials explored a parameter range of 0 to 128 presynaptic spikes and 0 to 512
postsynaptic synapses, in steps of 16 and 32 respectively. The former parameter controlled the number of DMA transfers to be made and processed, and the latter controlled the size of each transfer and the consequent cost of processing. In order to examine each stage of the pipeline, the simulated network was contrived to offer three modes of processing each spike: immediately discarding the spike; performing a DMA transfer of synaptic weights to data memory then discarding the transfer results; and performing a DMA transfer of synaptic weights to data memory then processing the transfer results into postsynaptic currents, as in a conventional simulation. Checks were performed to ensure that no packets were lost in transit between processors and that callback queues did not overflow. Callback priorities were chosen as simply as possible, such that the packet callback was priority minus one (preemptive) and all others were priority one (queueable); this ensured that packets were cleared from the communications controller immediately upon receipt and other callbacks were executed in the order in which their causative events occurred. The DMA callback was set to process the weights of the causative DMA transfer before setting up the next transfer; the term packet-processing pipeline should not be interpreted, as in the field of computer processor design, to mean that each stage was utilised in parallel. The processors were run at 200 megahertz and the router, system bus and off-chip memory were run at 133 megahertz.

A. Packet processing cost

The first stage of the packet-processing pipeline accepts a spike packet from the communications controller and looks up the address of the corresponding weights in off-chip memory; if none are found, the packet is simply discarded.

A projection was contrived between the spike source and target populations within which all synapse lookups failed immediately, thereby testing the first stage of the processing pipeline in isolation. Under these conditions, no excess packets were observed in any trial.

B. Direct memory access cost

In the second stage of the packet-processing pipeline, a DMA transfer is committed on the basis of the lookup results and the DMA engine copies weights while the processor is free to perform other computations.

A projection was created in order to commit a DMA transfer on receipt of a spike then dispose of the copy results on completion, thereby testing the first two stages of the processing pipeline in isolation. Figure 9 shows the worst observed excesses from all processors in each trial, superimposed with the constant product $xy = 10^4$ that represents the architectural target of ten thousand synaptic events per millisecond. DMA bandwidth is clearly sufficient for a single processor to work far in excess of the architectural target, and is almost sufficient to service fifteen processors in parallel.

Considering the latter case and the data point of twenty spikes innervating five hundred synapses, each of the fifteen processors retrieves ten thousand four-byte synapses per millisecond, which makes for a total throughput of six hundred megabytes per second. This observation of maximum DMA throughput is direct agreement with results obtained by Painkras et al. [7]. Note that excesses are considerably greater at the other end of the target curve: although the number of synaptic events under consideration is the same, the greater number of input spikes and DMA transfers makes for a significant increase in overhead.

Ultimately, it is clear that DMA bandwidth is a significant determinant of pipeline throughput. According to Painkras et al. each processor may retrieve around two hundred megabytes per second via DMA, so three processors operating at their maximum transfer rates will saturate the data bus. If four or more processors access the bus concurrently, they must share the available bandwidth and will therefore see a diminished throughput. Thus, the throughput observed in the right panel of figure 9 is the maximum achievable by the packet-processing pipeline.

C. Postsynaptic computation cost

In the final stage of the packet-processing pipeline, the copied weights are read iteratively and processed into postsynaptic currents for their target neurons.

A projection was created in order to commit a DMA transfer on receipt of a spike and then process the copied synaptic currents for their target neurons.
weights as normal, thereby testing the entire pipeline. Figure 9 again shows the worst packet excesses in comparison to the architectural target. Performance in both the single- and fifteen-processor cases drops to well below the curve, to almost exactly the same value. This suggests that postsynaptic computation so dominates the packet-processing pipeline that the cost of DMA transfers is negated.

A coarse estimate of processor utilisation during the millisecond simulation cycle is telling of the results obtained here. A processor running at 200 megahertz may execute 200,000 single-cycle instructions in a millisecond. According to the disassembled simulation code, each leaky integrate-and-fire neuron requires approximately 128 of these processor cycles and each synaptic event needs around 32. Simulation of 256 neurons consumes 30,000 cycles, which leaves approximately 170,000 cycles for handling synaptic events. As such, it should be possible to process around 5,000 synaptic events per millisecond; this curve is plotted in green on figure 9 and agrees with the first curve of the excesses remarkably well, barring the overhead-asymmetry.

The packet-processing pipeline throughput appears therefore to be defined entirely in terms of core thread performance. This explains why performance does not differ between the single- and fifteen-processor cases, but it raises the question as to why DMA-transfer time does not factor in the results. The DMA-transfer and postsynaptic-processing stages occur consecutively in the packet-processing pipeline, so the time to complete the former step must be vanishingly small if it is not to manifest in the performance data. Indeed, a transfer of \( n \) four-byte synapses at two hundred megabytes per second will take

\[
\frac{4nB}{2 \cdot 10^8 B/s} = 2n \cdot 10^{-8} s
\]

while processing the same quantity will take

\[
\frac{32nC}{2 \cdot 10^8 C/s} = 16n \cdot 10^{-8} s
\]

where \( C \) denotes processor cycles. As such, the DMA stage of the packet-processing pipeline takes only around a tenth of the total time, which is well within the error margins of the coarse utilisation estimates above. This explains why the performance of one processor, achieving the maximum DMA throughput, may be defined almost entirely in terms of processor utilisation. To explain the similar performance of the fifteen-processor case, it must be considered that only one tenth of each processor’s time is spent making DMA transfers; on average only two of the eighteen processors are making transfers at any time, so both see their full two-hundred-megabytes-per-second throughput without saturating the data bus. This argument boldly assumes that DMA transfers are uniformly distributed throughout the millisecond; this cannot be proven from the data but it is a likely interpretation of the results.

**D. Validation**

A final experiment was conducted to validate the results above. A single population of trial-varying size was con-figured such that each neuron fired at ten hertz, and initial membrane potentials were distributed so that firing was uniformly spread through time. The \( N \) neurons in each trial were split across \( \left\lfloor \frac{N}{2^{15}} \right\rfloor \) processors. A recurrent projection was made from the population onto itself with probability 0.1 and synaptic weight 0, so that received spikes would induce postsynaptic current computations but have no affect on the regular spiking. Each processor therefore computed

\[
N \cdot 256 \cdot p \cdot 10^{-3}
\]

synaptic events per millisecond.

Figure 11 shows the mean and variance in excesses across all processors as a function of population size \( N \). The blue line shows the expected number of packet excesses according to the results in figure 10. Although the standard deviations are large, the means of the observed results are in agreement with those expected, which suggests that the earlier experiments accurately model the packet-processing pipeline in simulations of recurrent networks.

**E. Execution time**

A complete SpiNNaker system consists of the core and DMA threads of each processor, which have been profiled above, and a host thread on a conventional desktop computer that constructs and controls simulations. As such, the total execution time of a simulation is a function of both host and chip performance. SpiNNaker simulations typically run in real-time, but the host machine may spend additional time generating and uploading data structures and downloading and processing results. The performance profile of the system as a whole, and its relation to comparable simulators, is therefore of significant interest.

The Vogels-Abbott benchmark is a network of excitatory and inhibitory neurons, much like that simulated in section II, that is conveniently encoded in a PyNN script by Brette et al. [1]. The benchmark was run on SpiNNaker and NEST with parameters ranging from \( 2^9 \) to \( 2^{13} \) neurons and \( 2^2 \) to \( 2^6 \) simulated seconds. The Python cProfile module was used to record the execution profile of each simulation. The host machine was based upon a triple-core AMD Athlon-II X3-445 processor clocked at three gigahertz served by
four gigabytes of memory; the SpiNNaker board contained four chips, totalling 72 processors, with processor clocks set to 200 megahertz and router, system bus and off-chip memory clocks set to 133 megahertz. SpiNNaker used one host processor to control simulations and 16 processors to execute them; NEST exploited just one host processor.

Figure 12 shows execution time of a 32-second simulation on SpiNNaker and NEST, as a function of neuron count. SpiNNaker beats NEST in all cases beyond 1,024 neurons. Run time on SpiNNaker remains a constant 32 seconds, and time to prepare data structures, load them and dump simulation results grows with model size. Figure 13 shows execution time of a 4,096-neuron model as a function of simulation time. Here, SpiNNaker beats NEST in all cases. Run time on SpiNNaker grows as the identity of simulation time, preparation and loading time remains constant, and dump time grows with simulation time.

The performance profile of SpiNNaker is easily explained. Simulations execute in real-time, so run time is consistently equal to the simulation period. Preparation and loading times are a quadratic function of model size; synapse structures comprise the majority of data to be generated and sent to the machine, and a recurrently connected network of size \( N \) with projection probability \( p \) forms \( pN^2 \) synapses. Dump time comprises the cost of retrieving and sorting spikes from the simulation, and is some function of model size, firing rate and run time. However, both the bandwidth and software exists for spikes to be dumped via Ethernet in parallel with the simulation, as shown by Sharp et al. [9, 8], so this runtime portion may be collapsed.

Performance comparisons between simulators are often complicated by vast differences in peak hardware throughput: achieving \( 2.5 \times \) speedup from an architecture that is \( 3 \times \) more powerful that its counterpart is rather unremarkable [6]. The sixteen application processors of a SpiNNaker chip have a combined total throughput of \( 3.2 \times 10^9 \) operations per second, which is similar to the \( 3 \times 10^9 \) operations per second available to the host processor. Naturally, there are innumerable differences in the hardware and software architectures of the two platforms, some favourable to one and some to the other, but neither excels purely in terms of arithmetic throughput. The 4,096-neuron simulations presented in figure 13, which use sixteen processors, are therefore a reasonably fair comparison of SpiNNaker and single-threaded NEST, although further experiments are required to compare the scaling properties of the simulators. Disregarding dump time, as justified above, the former achieves a sixfold speedup over the latter at all points beyond 4-second simulations.

F. Discussion

These experiments build a detailed performance profile of SpiNNaker simulations at both the processor and system levels. Results show that each processor may simulate 256 neurons with complex synaptic currents in real-time while processing 5,000 synaptic events per millisecond. A 32-second simulation of an 8,192-neuron network may be completed in less than 100 seconds, six times faster than the NEST reference simulator. A 64-second simulation of 4,096 neurons completes in a similar time with a similar speedup.

Examination of the packet-processing pipeline shows that the DMA thread can supply a processor with the 10,000 synapses per millisecond but that the core thread can handle only half this number. These data accurately predict the maximum processor throughput on at least one trajectory through the parameter space and clearly identify the pipeline bottleneck to be optimised. Unfortunately, the loop that dominates processor time is only 32 instructions long, so there is little room for improvement; careful programming may save three or four instructions and improve performance by one tenth to one fifth. Adjusting the API priorities and optimising the kernel routines may reduce overhead and improve performance in the many-spikes-few-synapses domain. However, each of these suggestions calls for significantly more programming and profiling, and easier gains may be had by working to exploit the abundant parallelism of the production SpiNNaker boards. Minute optimisations should not be considered until all processors are fully utilised.

The execution profile of the Vogels-Abbott benchmark shows that SpiNNaker outperforms NEST in all nontrivial simulations. Simulation of 4,096 neurons on SpiNNaker draws upon similar computational resources as the same simulation as NEST, yet the former shows a sixfold speedup...
over the latter. The SpiNNaker results demonstrate that load time becomes significant as models scale up, as predicted by Sharpt et al. [9], so high-bandwidth channels and data compression will be required to fuel larger simulations. Dump time increases likewise, but Sharpt et al. [8] show that results may be extracted from the machine in real-time, which comprehensively solves the problem. The NEST simulation kernel is written in C++, which the cPython is unable to report upon in detail, so further analysis of performance is not possible. However, the data is sufficient to show that SpiNNaker significantly accelerates simulation of models on the order of thousands of neurons. Furthermore, the trends in the data suggest that the performance gap between the two simulators will grow even more dramatically as models scale into tens of thousands of neurons.

IV. Conclusion

SpiNNaker is a novel architecture designed to simulate large-scale models of neural tissue correctly and quickly. This paper has presented experiments to show that SpiNNaker meets or approaches its architectural targets and is therefore an effective large-scale simulation platform.

SpiNNaker correctly simulates neural populations with recurrent projections and external input, although comparing results with reference simulators is not easy. There exists no published standard for the numerical solution of model differential equations, so no two simulators return precisely the same results in even the most simple simulations, as figure 1 shows. Nevertheless, it is demonstrated here that the outputs of SpiNNaker and NEST do significantly correlate in the established metrics for simulator comparison.

The performance profile of the simulation code shows that each processor may model 256 leaky integrate-and-fire neurons with exponential synaptic currents in real-time while processing 5,000 synaptic events per millisecond. The platform meets the performance goals set by Jin et al. [5] to simulate 1,024 Izhikevich neurons with delta synaptic currents, which are approximately one quarter as computationally expensive. However, Jin et al. set a target of processing 10,000 synaptic events per millisecond on the assumption that the processing loop would consist of approximately twenty instructions, whereas it actually contains nearer forty. Nevertheless, the profiling methodology developed here informs both subsequent code optimisations and correct load-balancing of simulations across processors.

The performance profile of the whole software stack shows that SpiNNaker significantly outperforms NEST using similar computational resources to simulate 4,096 neurons. Furthermore, the former simulator recruits additional processors and maintains a constant run time as model scale increases; only the load and dump times increase, and these may be ameliorated by on-chip generation and analysis of simulation data. These results suggest that SpiNNaker is an effective simulator of large-scale models of neural tissue, and thereby complement previous reports of the exceptional power-efficiency of the architecture [8].

REFERENCES


