The Deferred Event Model for Hardware-Oriented Spiking Neural Networks

Alexander Rast*, Xin Jin, Mukaram Khan, and Steve Furber

School of Computer Science, University of Manchester
Manchester, UK M13 9PL
{rasta,khanm,jinxa}@cs.man.ac.uk
{steve.furber}@manchester.ac.uk
http://www.cs.manchester.ac.uk/apt

Abstract. Real-time modelling of large neural systems places critical demands on the processing system's dynamic model. With spiking neural networks it is convenient to abstract each spike to a point event. In addition to the representational simplification, the event model confers the ability to defer state updates, if the model does not propagate the effects of the current event instantaneously. Using the SpiNNaker dedicated neural chip multiprocessor as an example system, we develop models for neural dynamics and synaptic learning that delay actual updates until the next input event while performing processing in background between events, using the difference between "electronic time" and "neural time" to achieve real-time performance. The model relaxes both local memory and update scheduling requirements to levels realistic for the hardware. The delayed-event model represents a useful way to recast the real-time updating problem into a question of time to the *next* event.

1 Real-time neural networks: the update timing challenge

Accurately modelling the continuous-time behaviour of large neural networks in real time presents a difficult computational choice: when to update the state? Conventional sequential digital processing usually prohibits real-time updates except on the largest, fastest computers, but dedicated parallel neural network hardware needs some time model. Broadly, two different architectures have become popular. One, the neuromorphic approach, e.g. [1], circumvents the state update problem altogether by using continuous-time analogue circuitry - at the price of hardwiring the model of computation into the system. The other, the parallel neurocomputer approach, e.g. [2] retains general-purpose digital processing but attempts to use a neural-specific, multiprocessor architecture - with some loss of speed and accuracy of state update. An ideal approach would combine the process abstraction capabilities of digital devices with the asynchronous-time model of analogue devices. SpiNNaker, a chip using event-driven asynchronous digital circuitry timed by its input rather than by an instruction clock, makes

* Corresponding author

this feasible for spiking neural networks having real axonal and synaptic delays, by abstracting a spike to an event. During the time between events, it is in effect "outside time", and can perform necessary background processing without affecting the model update. Having previously demonstrated a basic neural implementation on SpiNNaker, here we present a general method to maintain accurate temporal dynamics by deferring pending updates until the next input event. The method realises a solution to the update scheduling problem that suggests the possibility of practical large-scale, real-time neural simulation.

2 Deferred update dynamics

Real dynamic properties of neural networks allow us to relax the timing requirements dramatically. Most models, whether using temporal [3] or rate coding [4], [5], assume that the spike timing irrespective of shape determines the information coding. A typical active neuron fires at ∼10-20 Hz up to a maximum of ∼100 Hz [6]. Only a small number of a given population of neurons, typically ∼1-0.1%, will be active at any time, with 10% a reasonable upper limit. For a "typical" neuron containing 5000 dendritic connections with 1% activity, spiking at 10 Hz, we therefore expect an average input rate of 500 events (input spikes) per second, requiring an update rate of only 2ms. A worst-case situation: 100k inputs, 10% activity, 100 Hz firing rate, would require 1μs update rate. These are leisurely rates for typical digital processors running at hundreds of MHz. With real neurons having axonal delays, usually of the order of 1-20 ms, if the processor can propagate the required updates following an event in less time than the interval between events that affect a given output, it can use that time difference to defer the event processing until the occurrence of the *next* event. Simple time-domain multiplexing [7] is an established approach, but in addition, the processor can make use of the "dead space" to wait on contingent input events that may likewise affect the output. By performing temporal reordering of input events, it can ignore the fine-grained order of inputs. In Fig. 1, axons have finite delays. The processor schedules the updates, and the neuron can respond to further potential future input events with nonzero axonal delays, reordering them as necessary so that the neuron exhibits the proper dynamic behaviour even with nondeterministic input order. Delayed event processing thus allows more than time multiplexing; it decouples the temporal dynamics of the processing system from the temporal dynamics of the model.

3 Implementation of a neuron and synapse on SpiNNaker

3.1 The SpiNNaker hardware system

We have developed a universal spiking neural network chip multiprocessor, SpiNNaker (fig. 2), to support large scale real-time simulation [8]. 20 ARM968 processor cores with dedicated 64KB local memory implement neurons and their associated dendrites. A large off-chip SDRAM memory device stores synaptic data,

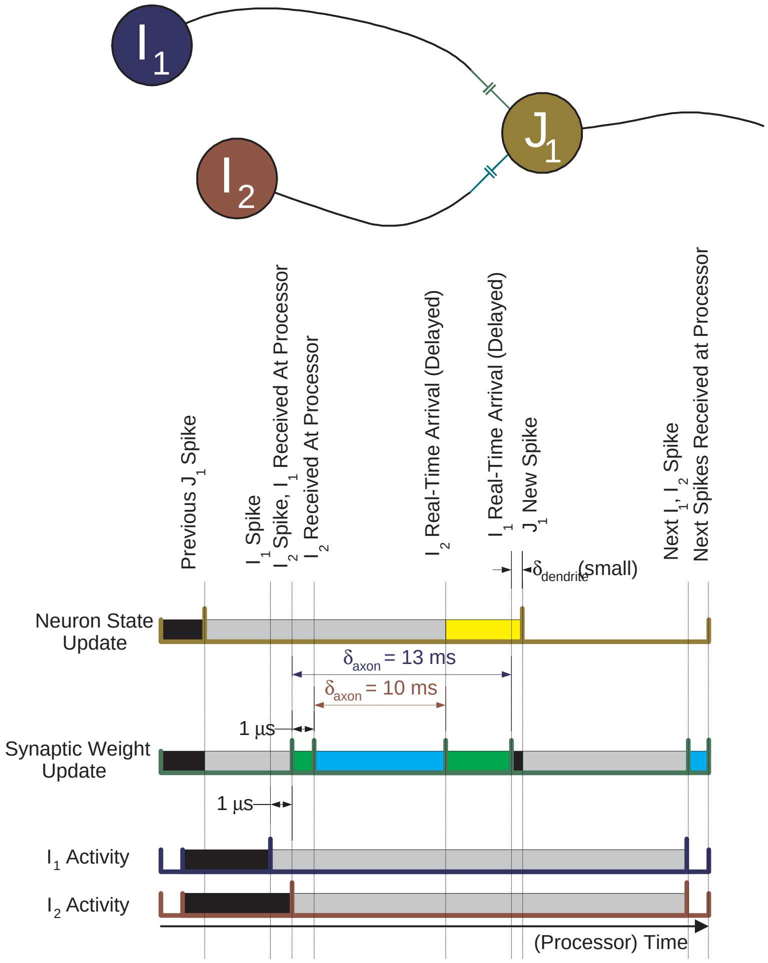

Fig. 1. Deferred Event Processing. At top is a very simple network, with schematic event spike trains at the bottom. The spike trains further show time before the current event (dark regions), time to process the current event (medium regions), and time when the processor can defer the processing (light regions). Neuron J_2 receives inputs from two other neurons I_1 and I_2 with delays of 13 ms and 10 ms respectively. Suppose that the processor performing the modelling can propagate the output in 1 μs. Neuron I_1 fires first in the model, with I_2 following close behind it after a 1 μs delay. Thus the corresponding events arrive at J_2 after 1 μs and 2 μs respectively. But because of the real delays in the model, the processor will not actually need the input from neuron I_1 for another 12.999 ms. Thus at 1 μs, it need do nothing with that input event other than record it. In particular, it can wait for input I_2, and when that event occurs at 2 μs, it can schedule its update event 9.999 ms into its future, with the update event for input I_1 still 12.998 ms in the future.

made to appear "virtually local" to the processor using an optimised DMA controller in concert with a 1Gb/s internal asynchronous Network on Chip (NoC) [9]. A configurable 6Gb/s per node global asynchronous packet-switching network, using an on-board source-based associative router together with an event-driven communications controller, implements axons and the associated connectivity map [10]. Using the "address-event representation" (AER) format [11], SpiNNaker signalling abstracts the neural action potential into a single discrete point event happening in zero time, a "spike". An AER packet simply records the source of the spike (and optionally an extra "payload" data field), to route it to target neurons potentially distributed over a system of up to 64K chips. Both the physical and temporal details of SpiNNaker hardware are invisible to the neural network model, allowing the device to be loaded and configured to support virtually any model of dynamics with any topology [12].

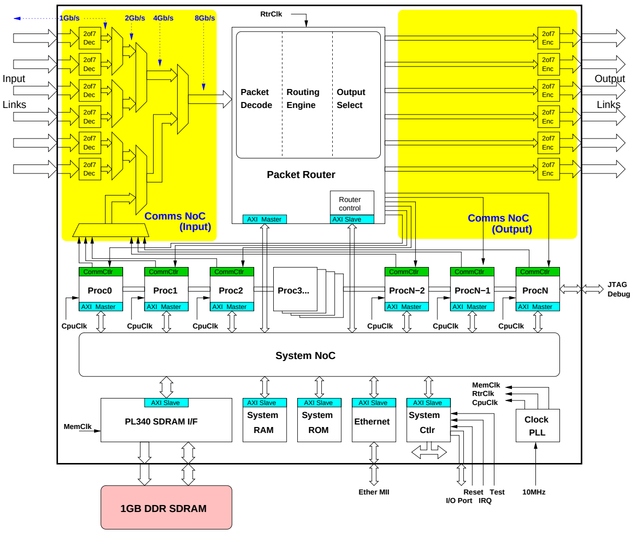

Fig. 2. SpiNNaker design, showing router, processors, and internal NoC components

3.2 Neuron

As a SpiNNaker case study, we have implemented the Izhikevich model [13] neuron because it is simple, instruction-efficient, and exhibits most of the dynamic properties of biological neurons. Since the ARM968 processor cores contain no floating-point support, and studies [14], [15], [16] indicate that 16-bit precision is adequate for most purposes, we represent all variables and parameters with 16-bit fixed-point variables. A 16-element array (fig. 3) represents input stimuli. Each element, representing a time bin having model axonal delay from 1 ms to 16 ms, carries a stimulus. When a spike packet with a delay of δ_m ms arrives at t_n ms, the processor increments the value in the $t_n+\delta_m$ ms time bin by the connection weight of the input synapse, deferring the neural state update until the global simulation time reaches $t_n+\delta_m$ - the time when the input "actually" arrives. During state update, we extract the accumulated stimulus for the current time bin from the data structure of one neuron and pass it along with the (static) parameters to the two Izhikevich equations, storing the new neural state values back to the data structure. For each update, we test the value of the membrane potential. If it reaches its reset value, the neuron fires and the processor will issue a multicast packet including the processor ID and the neuron ID. The processor maintains real-time accuracy as long as its neurons can be updated before shifting to the next bin position - nominally 1 ms intervals in our model.

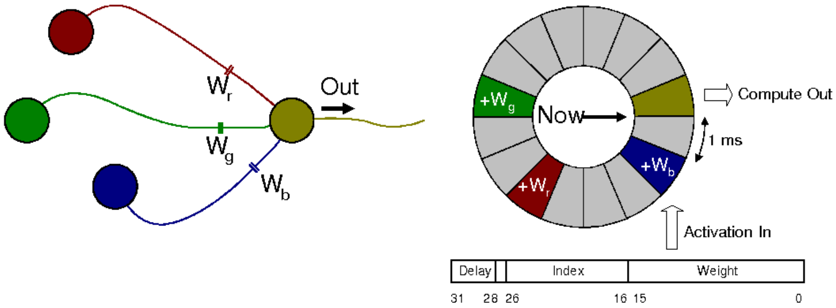

Fig. 3. SpiNNaker Neuron Binned Input Array. Inputs arrive into the bin corresponding to their respective delay. Output is computed from the bin pointed to by "Now".

3.3 Synapse

For the synapse, we have used the exponential spike-timing dependent plasticity (STDP) model [3]. Weight updates exploit the fact that the updated value will not be required until the input event *after* the current one. Each source neuron has an associated "row" in SDRAM containing a time stamp and target-indexed weight entries only for those target neurons on the local processor with nonzero weights (fig. 4), permitting a single table lookup per input event to retrieve the synapse data. For the time stamp we have used a 64 ms two-phase time window comparable to the $\sim$ 50 ms experimentally observed STDP time window [17], permitting 4 ms spike-time resolution within a 32K × 64ms coarse time period, enough to capture about 35 minutes without time rollover. With the arrival of an input, we update the time stamp and compute the *previous* weight update as follows: Retrieve the synapse row. For each connected neuron, compare its local coarse time stamp (postsynaptic spiking times) against the retrieved time stamp (presynaptic activation times). If the coarse time stamp matches, perform the weight update rule the model specifies, using the fine time stamp. (By coding the weight update as the power-of-2 bit-position in the fine time stamp, the processor can compute the update with a series of simple shift-and-add operations). If the neuron fires, update its local time stamp. If the delayed input time is within a synaptic window of the current time stamp, place a 1 in the stored (presynaptic) time stamp at the bit position corresponding to the time offset. Write back the updated synapses and time stamp to memory. The essential timing requirement is that the update computation takes less time than the time between inputs on the same synapse.

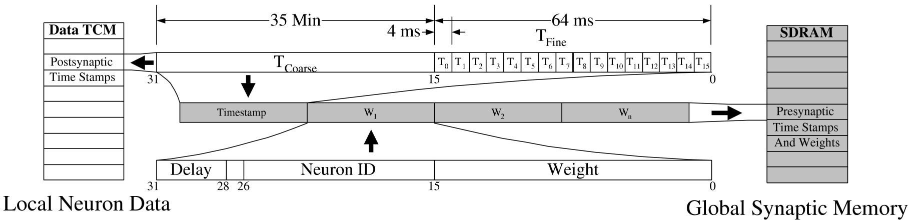

Fig. 4. Synapse Data Format. Each of the data words is 32 bits wide. Synapse rows in global memory have multiple weights and a time stamp word. Weights contain a 16-bit weight field and a 16-bit parameter field. In our experiments we used 4 bits of the parameter field to represent synaptic delays and 11 to index the target neuron. Each row in SDRAM has one time stamp (time of last presynaptic event), and each neuron in local memory a second (time of last postsynaptic spike). The time stamp has 2 fields, a fine and coarse time stamp. The fine time stamp is bit-mapped into 16 equal time segments with resolution $\frac{I_w}{16}$, where I_w is the total length of the time window, containing a 1 if there was an input activation at that real-time point within the interval. The coarse time stamp is an integer with a resolution of $\frac{I_w}{\phi}$, where ϕ is a (small) power-of-2 constant representing a time phase within the window I_w. The model updates this time stamp if a transmission event t_{new} occurs at $\mathrm{t}_{new} - \mathrm{t}_{last} \geq \frac{I_w}{\phi}$, where t_{last} is the current value of the coarse time step.

4 Application to system modelling

We have created a complete system-level simulation modelling multiple SpiNNaker chips using SystemC, and a cycle and ARM Instruction Set accurate simulation using ARM SoC designer, to verify the design, support early application development, and test the overall chip/system behaviour. We calibrated cycle-accurate SystemC behaviour of modelled chip components against their Verilog-based simulations used for chip fabrication. We performed case studies to test the functionality of the chip, the routing fabric, and the system under various scenarios. To verify the delayed-event model we implemented the Izhikevich [13] neural dynamic model using Xin Jin, et al.'s scheme [14] and successfully tested a small population of neurons (fig. 5). We tested the physical delays of the neurons in the population to determine the time margin a typical input would have to complete its processing (fig. 5). According to our simulations the update time for one neuron is about 240 ns. With 1000 neurons, this corresponds to a total update time of 0.24 ms - comfortably within the worst-case 1 ms delay margin even with nondeterministic arrival times. By delaying the state update, we effectively compress these computations into the time immediately after an update event, releasing the slack for further processes such as synaptic updating.

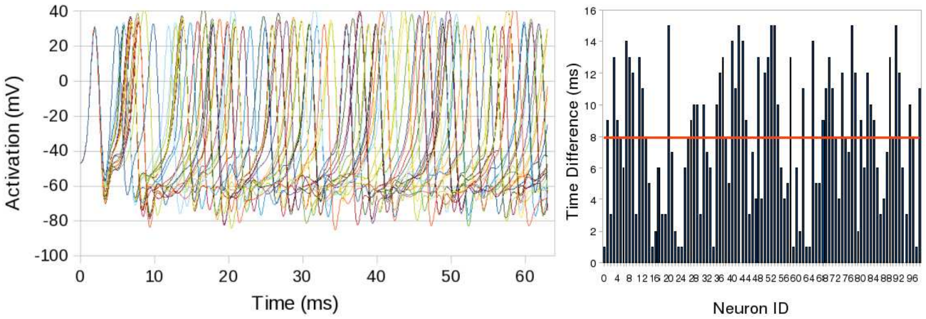

Fig. 5. Neuron spike traces (L) and delay margins (R). Traces are the output response to delayed inputs. The bars show the time difference between the modelled axonal delay and the hardware delay, measured from spike packet transmission time to packet reception time. Worst-case delay margin is 1 ms. Mean delay margin is 7.93 ms.

5 Conclusions

We have developed a model capable of achieving biologically realistic real-time performance on event-driven neural network hardware, with nondeterministic signal timing. The deferred-update model permits not only real-time signal timing resolution but also more efficient processor utilisation, since each processor can schedule its updates according to the model-time rather than the hardware-time event sequence. SpiNNaker has three dimensions of configurability, letting the user specify the topology, processing dynamics, and temporal dynamics according to the needs of the model, rather than those of the hardware. Such a "neuromimetic" system mitigates both against instant hardware obsolescence and the need to make hard decisions about what neural models to experiment with in view of the hardware available. It is, perhaps, therefore a more viable architecture for future neural systems than approaches that impose hard physical constraints. Our hope is that SpiNNaker might encourage dialogue between the biological scientists and the computer scientists, to link phenomenological observations to behavioural models of computation. Such systems could promote a two-way flow of learning, discovering on the one hand more about how the brain functions and on the other new and alternative computing models.

Acknowledgements We would like to thank the Engineering and Physical Sciences Research Council (EPSRC), Silistix, and ARM for support of this research. S.B. Furber is the recipient of a Royal Society Wolfson Merit Award.

References

1. Indiveri, G., Chicca, E., Douglas, R.: A VLSI Array of Low-Power Spiking Neurons and Bistable Synapses With Spike-Timing Dependent Plasticity. IEEE Trans. Neural Networks **17**(1) (January 2006) 211–221
2. Mehrtash, N., Jung, D., Hellmich, H., Schönauer, T., Lu, V.T., Klar, H.: Synaptic Plasticity in Spiking Neural Networks (SP^{2}INN): a System Approach. IEEE Trans. Neural Networks **14**(5) (September 2003) 980–992
3. Gerstner, W., Kempter, R., van Hemmen, J.L., Wagner, H.: A Neuronal Learning Rule for Sub-millisecond Temporal Coding. Nature **383**(6595) (Sep. 1996) 76–78
4. Shadlen, M.N., Newsome, W.T.: The Variable Discharge of Cortical Neurons: Implications for Connectivity, Computation, and Information Coding. J. Neuroscience **18**(10) (May 1998) 3870–3896
5. Masuda, N., Aihara, K.: Duality of Rate Coding and Temporal Coding in Multilayered Feedfoward Networks. Neural Computation **15**(1) (January 2003) 103–125
6. Dayan, P., Abbott, L.: Theoretical Neuroscience. MIT Press, Cambridge (2001)
7. Goldberg, D., Cauwenberghs, G., Andreou, A.: Analog VLSI Spiking Neural Network With Address Domain Probabilistic Synapses. In: Proc. 2001 Int'l Symp. Circuits and Systems (ISCAS2001). (2001) 241–244
8. Furber, S.B., Temple, S.: Neural Systems Engineering. J. Roy. Soc. Interface **4**(13) (April 2007) 193–206
9. Rast, A., Yang, S., Khan, M.M., Furber, S.: Virtual Synaptic Interconnect Using an Asynchronous Network-on-Chip. In: Proc. 2008 Int'l Joint Conf. Neural Networks (IJCNN2008). (2008) 2727–2734
10. Plana, L.A., Furber, S.B., Temple, S., Khan, M.M., Shi, Y., Wu, J., Yang, S.: A GALS Infrastructure for a Massively Parallel Multiprocessor. IEEE Design & Test of Computers **24**(5) (Sep.-Oct. 2007) 454–463
11. Lazzaro, J., Wawrzynek, J., Mahowald, M., Silviotti, M., Gillespie, D.: Silicon Auditory Processors as Computer Peripherals. IEEE Trans. Neural Networks **4**(3) (May 1993) 523–528
12. Khan, M.M., Lester, D., Plana, L., Rast, A., Jin, X., Painkras, E., Furber, S.: SpiNNaker: Mapping Neural Networks Onto a Massively-Parallel Chip Multiprocessor. In: Proc. 2008 Int'l Joint Conf. Neural Networks (IJCNN2008). (2008) 2849–2856
13. Izhikevich, E.: Simple Model of Spiking Neurons. IEEE Trans. Neural Networks **14** (November 2003) 1569–1572
14. Jin, X., Furber, S., Woods, J.: Efficient Modelling of Spiking Neural Networks on a Scalable Chip Multiprocessor. In: Proc. 2008 Int'l Joint Conf. Neural Networks (IJCNN2008). (2008) 2812–2819
15. Wang, H.P., Chicca, E., Indiveri, G., Sejnowski, T.J.: Reliable Computation in Noisy Backgrounds Using Real-Time Neuromorphic Hardware. In: Proc. 2007 IEEE Biomedical Circuits and Systems Conf. (BIOCAS2007). (2007) 71–74
16. Daud, T., Duong, T., Tran, M., Langenbacher, H., Thakoor, A.: High Resolution Synaptic Weights and Hardware-in-the-Loop Learning. In: Proc. SPIE - Int'l Soc. Optical Engineering. Volume 2424. (1995) 489–500
17. Markram, H., Tsodyks, P.: Redistribution of Synaptic Efficacy Between Neocortical Pyramidal Neurons. Nature **382**(6594) (August 1996) 807–810