A Hardware/Application Overlay Model for Large-Scale Neuromorphic Simulation

Alexander Rast
Oxford Brookes University
Oxford, UK
ORCID ID 0000-0001-9934-7191

Mahyar Shahsavari
Imperial College London
London, UK
ORCID ID 0000-0001-7703-6835

Graeme M. Bragg
University of Southampton
Southampton, UK
ORCID ID 0000-0002-5201-7977

Mark L. Vousden
University of Southampton
Southampton, UK
ORCID ID 0000-0002-6552-5831

David Thomas
Imperial College London
London, UK
ORCID ID 0000-0002-9671-0917

Andrew Brown
University of Southampton
Southampton, UK
ORCID ID 0000-0002-0700-9433

***Abstract*—Neuromorphic computing is gaining momentum as an alternative hardware platform for large-scale neural simulation. However, with several major devices and systems available and planned, often with very different characteristics, it is not always clear which platform is suitable for which application. Simulating the platform on conventional computers is typically too slow to be of use, but an alternative approach is to implement an 'emulation' of the hardware in FPGAs which can execute at near-hardware speeds but does not commit to a specific hardware architecture. We present an *overlay* model - a method which superimposes bespoke features on top of a standard template - in both hardware and software to implement neuromorphic architectures using the POETS (Partially Ordered Event Triggered Systems) system. This combination of overlays permits very large-scale simulations to be performed in real time for hardware exploration or application verification, while retaining the flexibility to redefine either the hardware or software layer, if results indicate potential to improve performance, or significant design problems. Using this system we simulate up to 500,000 neurons on a single-box system, that can be scaled to ∼4,000,000 neurons in an 8-box configuration. Results indicate the crucial constraint for real-time simulation: peak input spike rate per neuron; and help to optimise both hardware and software around neural application requirements. The preliminary architecture demonstrates the feasibility of an overlay model, while indicating directions for future neuromorphic systems. With POETS, we introduce a platform that can help to shape and investigate the neuromorphic architectures of the future.**

***Index Terms*—neuromorphic, parallel, spiking, event-based**

I. Introduction

NEUROMORPHIC chips are devices that implement spiking neural networks directly in hardware. Such devices offer unique hardware capabilities, particularly attractive for embedded and robotic systems. Advances in both scope and scale of neuromorphic designs have made them appear to be one of the most promising new computing architectures of the forthcoming decade. However, a potential limiting factor in adoption is that such devices can be nondeterministic or subject to process variation, and traffic patterns across the chip may depend heavily upon the implemented network. As neuromorphic chips become larger, not only will design space exploration prior to fabrication become crucial, it will also be important to be able to simulate and explore the dynamic characteristics of the system in real-world applications. Pure software simulation is too slow for this purpose and cannot usually interface with, e.g. sensors that may be providing real-time data. It is therefore desirable to have a hardware platform that abstracts as much of the neuromorphic design as necessary to make sense of the internal operation, whilst retaining enough of the dynamic features to be usable in a real-time context. FPGAs seem like an obvious candidate for such a plaform. We have created an FPGA-based system, POETS, based on a series of hardware and software overlays that can simulate large-scale networks of spiking neurons with 2 important advantages:

- the ability to experiment with design ideas without having to commit to expensive custom silicon.
- scaling properties that permit networks to be simulated at real-world speed and scale.

This system represents an intermediate between hardware neuromorphics and software simulation: a prototyping platform to facilitate design space exploration at large scales for neuromorphic systems.

This work is supported by EPSRC grant EP/N031768/1.

II. Neuromorphic Platforms

A. 'Classical' Neuromorphic Chips

Neuromorphic computing has its origins in the work of Carver Mead's group in the early 1990's developing analogue hardware implementations of neural networks thought to replicate the physics of neurons directly in silicon [1]. Initially small-scale, these devices have grown in both size (in terms of number of neurons or number of connections) and features (e.g. on-chip learning [2] and multi-model support [3]) Modern large-scale analogue neuromorphic systems are capable of implementing many thousands of neurons [4], often with several choices of neuron and synapse model [5]. Analogue neuromorphics offer two important advantages: direct implementation

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

of potentially complex exponential and multiplication functions, and very low power consumption. On the other hand, historically, such devices have encountered challenges with device variability [6], precise implementation of particular dynamics [7], hardware interfacing, and scaling to very large sizes (hampered by analogue process technology constraints). It is possible to build functional analogue neuromorphic systems [8], but locating and matching on-chip devices with desired transfer characteristics for a given problem may be hard [9], and there are inevitably model compromises that make such chips less attractive for computational neuroscience than had been originally hoped.

B. Digital Neuromorphic Devices

Faced with these issues, and also with the vexing problem of toolchain support with often nonstandard devices, many researchers have turned to digital technology to implement neuromorphic-like devices [10]. Some early attempts simply co-opted more general-purpose parallel processing platforms and implemented neural networks as an application overlay on the hardware [11]. At around the same time as Mead's pioneering work, another community was exploring hardware implementations of classical machine-learning neural networks [12]. While not strictly 'neuromorphic', the recent explosion of discoveries in deep learning has revived interest in such devices, with several convolutional neural network chips available today giving various levels of acceleration [13], [14].

By contrast, in the mid-2000s, a new style of digital neuromorphic chip emerged using low-power embedded cores connected via a network-on-chip [15]. The two leading examples of this approach are the SpiNNaker system from the University of Manchester [16] and IBM's TrueNorth [17]. These chips exemplify different choices of operating regime and design intent: SpiNNaker features a flexible internal architecture using industry-standard ARM cores to allow for wide choice of neural model and simulation approach, while the IBM system uses fixed-function cores implementing specific neural processing functions which can be configured much like an FPGA to achieve a given application. Recently, the work of Eliasmith, et al. in the Neural Engineering Framework [18] has been implemented using a hardware accelerator [19], which in some respects represents a hybrid of the two approaches.

However, it is an open question which architecture is optimal in a given application, and there is a complex tradeoff between local serialisation vs. global parallelism which has revealed many issues but thus far few definitive solutions [20]. There is a need for design-space exploration for future such devices, but traditional hardware platforms are too slow to implement them effectively, and thus to date, much of digital neuromorphic design has been by ad-hoc choices made using 'back-of-the-napkin' calculations or average-case estimates of packet traffic, processing time, and numerical range.

C. Alternative Platforms

Finally, there is a group of quasi-neuromorphic platforms at various stages of development which either leverage commercially available parallel hardware, or advanced technologies that may yield larger-scale designs in future. The memristor is the quintessential representative of the latter with several designs proposed or implemented, although systems remain at small scale; this is a research technology for the future [21]. Several groups have explored using GPUs as a neural computing platform, with some success, albeit requiring very careful mapping of network to the target architecture [22]. A large community has been using FPGAs as a convenient platform either for large-scale implementation [23] or design space exploration [24]. FPGAs are particularly attractive because they provide many of the underlying architectural facilities of neuromorphics, benefit from the latest process technology advances, run at hardware speed without the development commitment of a full-custom ASIC [25], and can always be reused for a different project or design attempt if the initial design does not work out as expected. Notwithstanding, FPGAs present a major challenge to neuromorphic implementations: an extremely generic hardware substrate which does not yield immediately clear design choices. What would be ideal would be a standardised configuration for an FPGA or digital device that implements the underlying hardware model of neuromorphics but facilitates design space exploration - with the capability to scale to (and simulate) large scale networks without imposing overly restrictive model assumptions.

III. The POETS overlay

POETS describes a general, abstract hardware model that could be implemented in various ways, with the following specific properties amenable to a representation as a graph:

Self-contained compute elements A POETS platform consists of a number of possibly heterogeneous processors whose execution is entirely independent and have no explicit timing relation.

Processor hierarchy POETS defines a system with 'worker', 'supervisor', and 'executive' processors having successively greater levels of global visibility and operating system support.

Message-based interprocess communication A message: a 64-byte data item with application-defined contents, is the atomic unit of communication.

Local memory Each processor has private access to its own memory. There are no shared memory regions, and no coherency mechanisms in the hardware.

Guaranteed message delivery Once a message has been issued from a given processor, it is guaranteed to reach its target in finite time.

Asynchronous message timing Delivery is guaranteed, but messages may arrive out of order and at any time.

A. The POETS hardware system - background

The POETS hardware model is designed to implement a scalable many-core platform using very small, simple cores. This favours large numbers of simple devices with low computing complexity and high parallelism - an ideal case for neural modelling.

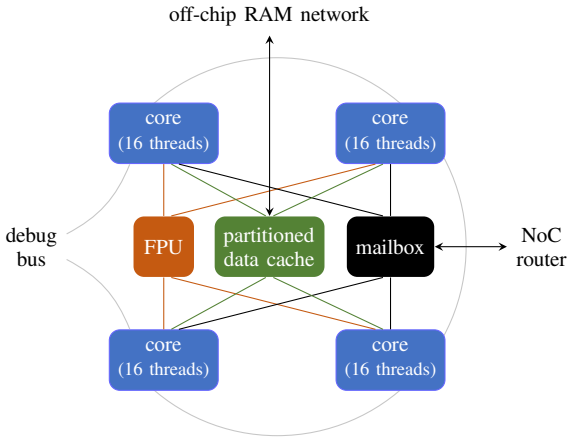


Fig. 1: Default configuration of a Tinsel tile, in which four cores share a floating-point unit, a mailbox, and a data cache. A tile connects to its surrounding context via the debug bus, the off-chip RAM network, and the network-on-chip.

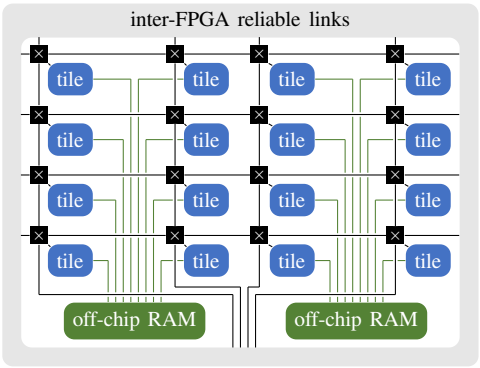


Fig. 2: Default configuration of the Tinsel NoC on a single FPGA. Tiles are connected together by dimension-ordered routers, and inter-FPGA links (4× 10G SFP+) are connected to the NoC rim. Each off-chip RAM block contains a DDR3 controller and two 8MB QDRII+ SRAM controllers.

In [26] we have introduced the details of the current implementation - called Tinsel. Tinsel is a highly-parameterised manycore design with a regular structure, consisting of a scalable grid of *tiles* connected by a reliable communication fabric (Fig. 2). It can be configured for various architectural tradeoffs. The *default configuration* is currently implemented on an 8-box cluster, each with 6 DE5-Net FPGAs supporting 64 cores. The comms fabric extends both within each FPGA and throughout the cluster. A tile, depicted in Figure 1, contains one or more instances of each of the following:

Core This is a custom 32-bit floating-point-enabled core implementing a subset of the RV32IMF profile of the RISC-V instruction set. It is a heavily-threaded barrel-scheduled core (context switching on every clock cycle), implementing 16 hardware threads by default.

Mailbox A mailbox contains a memory-mapped scratchpad that stores, by default, up to 1KB of incoming and outgoing messages for each thread that it serves. The mailbox allows threads to trigger transmission of outgoing messages, to allocate space for incoming messages, and to consume those messages when they arrive.

Data Cache This is an 8-way set-associative writeback cache that optimises access to the large off-chip memories available on each FPGA board. Communication via shared memory is unsupported in POETS, so we partition the cache by thread id (threads do not share cache lines).

FPU Each Tinsel tile implements the 'F' part of the RV32IMF profile via a 32-bit FPU shared between all the cores on the tile. However, to compare with existing hardware, we use a neural model that does not require floating-point processing.

B. The general POETS application model

The POETS application model describes a processing abstraction independent of the physical hardware implementation. One of the important tasks in developing a neural simulation (or, indeed, any application) on POETS, therefore, is mapping the problem to the application model.

POETS assumes an application can be described in terms of a graph of vertices and edges. Vertices, called *devices*, encapsulate some simple atomic unit of processing, *edges* carry event-based information (asynchronously) between devices. A graph-based application ('Graph Schema') transforms the abstract representation into an XML specification containing a GraphType and a GraphInstance section.The general mapping approach decomposes the graph into clusters of devices dominated by intra-cluster edges over inter-cluster edges. The POETS configuration system, called the 'Orchestrator', handles the intermediate translation from high-level specification to machine-executable code (Fig. 3).

A device is a purely event-driven process that responds via *handlers* to *messages* delivered over edges. An event is triggered, and a handler executed, when one of the following 4 conditions occurs:

- A message arrives on an input edge (OnReceive).
- A message is sent on an output edge (OnSend).
- A change in internal state, as a result of another event, makes a particular output edge ready to send a message (OnRTS).
- None of the other conditions has occurred, but the device has reached an 'interesting' state (OnIdle).

To conserve power and avoid unnecessary spinning, OnIdle exits with a result that indicates whether something 'interesting' happened that should trigger further events (such as OnRTS). If it exits without such a flag, the system waits for the next real event and does not execute OnIdle.

Devices and edges may have internal state, split into 2 components: *properties*, fixed over the lifetime of the application; and *state*, that may vary over the lifetime of an application. A given event may have only 2 effects: it can change the state, and/or it can cause a message to be sent. Messages, state, and properties all have an associated *type* that describes their data layout. Messages have a fixed, small maximum size (currently

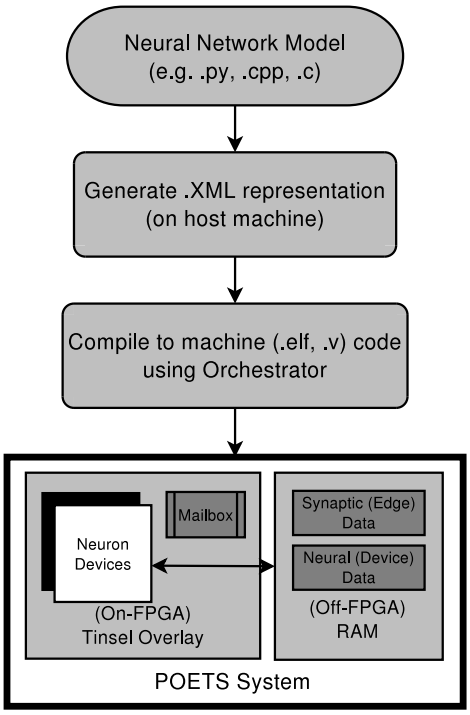


Fig. 3: Translation of the high-level software representation of the network to the low-level implementation on the POETS hardware.

64 bytes). Properties and State may have unlimited-size types, but the assumption is that they are small and simple types.

The actual implementation of devices uses a boilerplate code template, the *Softswitch*. In abstract form, a softswitch is nothing more than a polled event loop with an attached data structure. Most of the event loop and the data structure is predefined; the specific application is an overlay that replaces stub sections: the handlers and internal state pointers, with actual data values or code. The GraphType section in the XML specification contains the additional data definitions and code fragments necessary to define the application overlay, while the GraphInstance section generates individual devices and edges. A single softswitch normally resides on a thread and may instantiate multiple devices; if it does, the devices within it are serialised, although softswitches execute concurrently.

C. The neuromorphic overlay

We have developed a neuron node based on four events to handle fan-in, fan-out, clock, and state update. As a first model implementation, we have used an approach inspired by the comparable SpiNNaker system. This model is based on a fixed-timestep ODE (Ordinary Differential Equation) solver and a target-annotated delay applied to synaptic inputs. Such a model assumes the following:

- Hardware delays are trivial compared to (simulated) model delays.
- All devices can complete their timestep update within the timer interval.
- Synaptic delays are in a fixed, discrete, nonzero range.
- Spikes are sent immediately upon exceeding threshold.

A further set of assumptions are not fundamental, but establish the intended regime of operation of the model:

- Simulated neurons execute in wall-clock real time.
- All values are representable as 64-bit fixed-point numbers.
- Maximum input fan-in per device (neuron) is ~ 1000.
- Number of input spikes/device/time tick is small (~ 10).
- Only a small subset of neurons (~ 1000) need to be recorded externally.

1) Synaptic injection: Synapses inject a value (whether current or conductance) into the neural model after a certain synaptic delay. To account for different delays, a neuron contains a circular buffer of aggregate synaptic injections with one slot for each supported delay value. Each delay is an offset from a buffer pointer, incremented as time progresses, to the slot associated with 'now'. Inputs to synapses trigger buffer updates depending upon the synapse type, one of:

'Current Jump' synapse type: This, the simplest rule, simply injects the current value in state directly into the bin at the offset position.

Exponential Current synapse type: The synapse's value is treated as a peak injection at the offset position, from which the handler computes values for successive bins by exponentially decaying the injection according to a term

$$\tau_s \frac{dI_{syn}}{dt} = -I_{syn}$$

up to the point where either all available bins have an injection value, or the residual current is below the representational precision of the current bin.

Exponential Conductance synapse type: This type works like the exponential current type, only the value looked up for the edge is a conductance g_s varying in the same way:

$$\tau_s \frac{dg_s}{dt} = -g_s$$

whose injection will be determined when the neuron state update is performed (and generally depends upon it). Synapse types cannot generally be mixed in a given device (neuron).

2) Simulation timer: The model requires a timer to drive the simulation. Therefore, the POETS system must generate a regular source of timing events. To implement this, we use the OnIdle event in combination with a hardware feature, TinselCycleCount(), which returns the local clock count as a 32-bit wrapping unsigned integer. To get a desired timing value, we implement an internal counter within each device and perform a modulo-timer computation on the cycle count. Although each device executes its own timer, the time step is considered a global property of the simulation.

3) Event handlers: Processing for each of the 4 events is as follows:

OnReceive: This is normally the highest-priority event. When a spike arrives, OnReceive first performs any STDP update, if enabled, then injects the appropriate value(s) into the circular buffers according to the method described in III-C1.

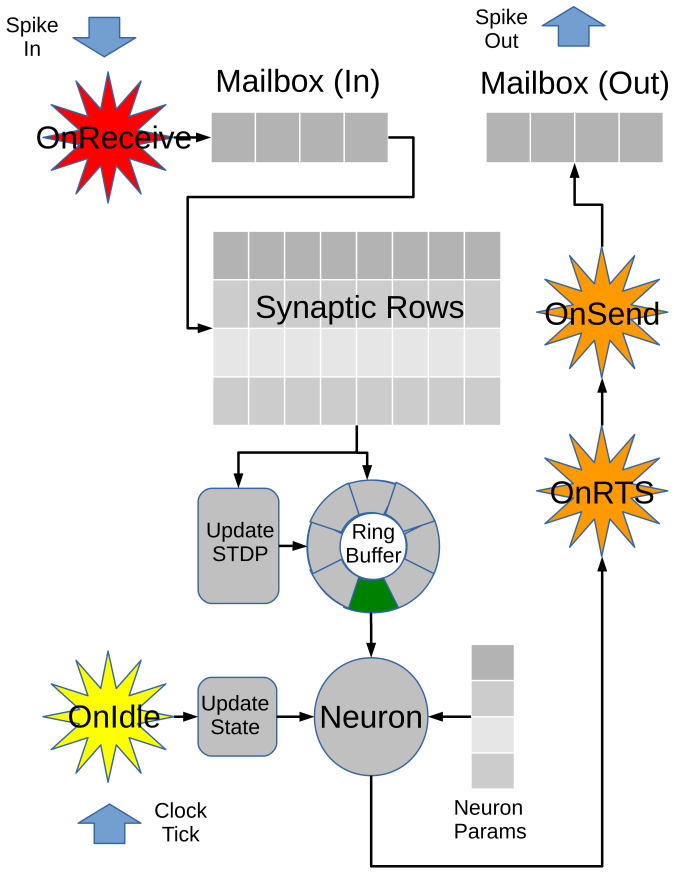


Fig. 4: Implementation of the POETS neural model overlay. The 4 main events trigger processing as described in III-C3. Maiboxes automatically send and receive messages through the hardware overlay. Synaptic rows are implemented as Edge State. Neural parameters are Device Properties. The STDP update is optional.

OnRTS: This event is executed immediately after any On-Receive, OnSend, or OnIdle event that yields an 'interesting' result. It simply detects if the critical neuron state value (typically, the voltage) is above spiking threshold (in device properties), then raises a ReadyToSend flag if it is.

OnSend: If the ReadyToSend flag is set, this event will reset state values to their 'reset' value (in device Properties), and if the synaptic type is one of the Current types, and a refractory period is specified, will set the refractory period counter to the number of time steps specified in τ_f. OnSend does not issue the spike explicitly; this is automatically performed by the generic softswitch machinery.

OnIdle: This first checks the timer as specified in III-C2. If the cycle count modulo-timer-interval is 0 (which can be done by simple masking operations, no division is required), OnIdle updates the neural state. In the case of current-type synapses the value in the current bins is used explicitly wherever I_{syn}, the synaptic current, is included in the computation. For conductance-type synapses, the value to be used to update voltage-dependent terms is computed as

$$\frac{dI_{syn}}{dt} = G_t(V - V_{rev})$$

where I_{syn} is the synaptic input current, G_t is the aggregate conductance value in the bin at time step t, V is the membrane voltage, and V_{rev} is the synaptic reversal potential.

IV. Neural Simulation Results

A. Spiking Neural Model

1) Neurodynamic details: What we have described thus far is the general model - the overlay - for spiking neural simulation. This can accommodate a wide variety of neural types. Some model parameters are global to the entire simulation and are stored as Graph Properties, as follows:

Variable	Meaning	Type	Units
t_{end}	End Time	`uint32_t`	ms
t_{tick}	Clocks Per Tick	`uint32_t`	4ns
$log_2\tau$	Time Scaling Factor	`int8_t`	log_2Hz
τ_{STDP}	STDP Window Size	`uint32_t`	ms
STDP_En	Enable STDP	`uint8_t (bool)`	-

For preliminary testing, we chose 2 models: the leaky-integrate and-fire (LIF) neuron and the Izhikevich neuron, with fixed, current-based synapses (no STDP). The basic LIF equation is:

$$\tau_m \frac{dV}{dt} = V_s - V + RI_{syn}$$

with the auxiliary conditions

$$\text{if } V \geq \theta, \; V \to V_r$$

and

$$\text{if } t - t_{last} < \tau_f, V = V_r$$

This form specifies an LIF neuron with time constant τ_m voltage threshold θ, membrane resistance R, rest voltage V_s, reset voltage V_r and refractory period τ_f. I_{syn} is the total synaptic current input. The model implements a simple form of refractoriness, clamping the voltage at reset until the refractory period is finished (the specified period is the time of absolute refractoriness, from which point the neuron is in a relative refractory state until the voltage relaxes from V_r to V_s). t_{last} indicates the time of the last spike.

The 'standard' Izhikevich form [27] is

$$\left.\begin{aligned} \frac{dV}{dt} &= 140 + (5 + 0.004V)V - U + I_{syn} \\ \frac{dU}{dt} &= a(bV - U) \end{aligned}\right\} \quad (1)$$

with the auxiliary conditions

$$\text{if } V \geq \theta, \left.\begin{aligned} V &\to c, \\ U &\to U + d \end{aligned}\right\} \quad (2)$$

As usual, a, b, c and d in the Izhikevich model are purely numeric parameters which tune the model for a particular

spiking behaviour. U is a relaxation term that sets the rate at which the model relaxes back to equilibrium.

Both models have the following common variables for neural state (Device State) and parameters (Device Properties):

State:

Variable	Meaning	Type	Units
V	Membrane Voltage	`int32_t`	μV
$I_{syn}R$	Injection Buffer	`int32_t[16]`	μV
ϕ	Circular Buffer Offset	`uint8_t`	-
t	Simulation Time	`uint32_t`	ms
t_{spikes}	Spike Times Bitmap	`uint32_t`	-
t_{last}	Last Spike Time	`uint32_t`	ms
Clk_0	Start Clock	`uint32_t`	4ns

Properties:

Variable	Meaning	Type	Units
θ	Threshold Voltage	`int32_t`	μV
$V_r(c)$	Reset Voltage	`int32_t`	μV
id	Neuron ID	`uint32_t`	-
record	Record neuron?	`uint8_t (bool)`	-

with the LIF model introducing the following additions:

Properties (LIF):

Variable	Meaning	Type	Units
V_s	Rest Voltage	`int32_t`	μV
ω	Characteristic Frequency	`uint32_t`	Hz
τ_f	Refractory Period	`uint32_t`	tick bitmask

and the Izhikevich model the following:

State (Izhikevich):

Variable	Meaning	Type	Units
U	Relaxation Variable	`int32_t`	μV

Properties (Izhikevich):

Variable	Meaning	Type	Units
a	Relaxation Gain	`int32_t`	-
b	Relaxation Scaling	`int32_t`	-
d	Relaxation Reset Offset	`int32_t`	μV

For computational simplicity, especially avoiding divisions and excessive multiplications, we transform some of the variables: τ_m is inverted to produce ω, and the fixed membrane resistance R is multiplied by the synaptic current I_{syn} and stored directly in synaptic state (In the Izhikevich model this is built into the equation itself). Synapses (edges) have the following Edge State and Edge Properties:

State:

Variable	Meaning	Type	Units
$I^i_{syn}R$	Synaptic Current	`int32_t`	μV
t^i_{spikes}	Presynaptic Spike Times Bitmap	`uint32_t`	-
t^i_{last}	Last Presynaptic Spike Time	`uint32_t`	ms

Properties:

Variable	Meaning	Type	Units
δ	Synaptic Delay	`uint8_t`	ms
ω_{Pot}	Potentiating Decay Frequency	`uint32_t`	Hz
ω_{Dep}	Depressing Decay Frequency	`uint32_t`	Hz
A_{Pot}	Potentiating Weight Increment	`int32_t`	μV
A_{Dep}	Depressing Weight Decrement	`int32_t`	μV

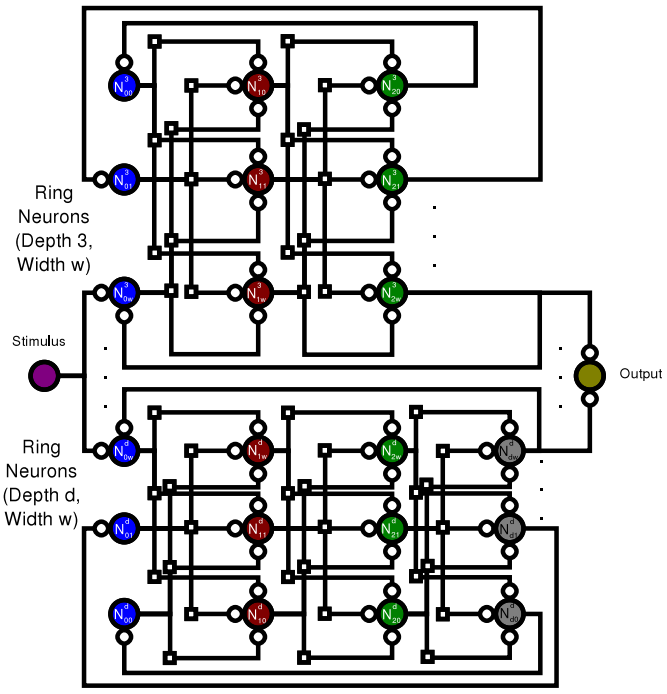


Fig. 5: Synfire ring network for testing. The network consists of rings of distinct depths (of 3, 5, 7, 11, 13, 17, and 19 layers) and widths (arbitrary). Open circles indicate (potential) synapses. Open squares indicate probabilistic connections.

For static synapses, only the delay and synaptic current are used, other variables relate to STDP parameters. (These additional variables are optimised out by the compiler at instantiation time if STDP is not enabled).

2) Network model: To test the functionality of the model we have created a network designed in the limit to be a 'model breaker': one with dynamics that stress test the hardware and software overlay to the maximum. The model we created (Fig. 5) is inspired by the 'synfire chain' spiking networks and follows the general pattern of the model in [28]. To recap this model, it consists of a number of rings of neural populations (pools), of varying ring lengths and pool sizes, with inter-pool connectivity set by a probabilistic connection parameter that ensures no ring has an 'open link' with no connections between a population and the successive one in the ring. All of the rings connect at one stage to an output neuron, whose threshold is set to fire only when all its inputs fire synchronously (in the same time tick). The appearance of a spike at the output indicates that the network is functional. Since by design, spikes from each population are issued synchronously to neurons in the successive population, each receiving neuron will receive a worst-case event rate: all synapses will become active in the same timer tick. This pattern of highly synchronous firings is the most challenging case for the simulator to handle, whilst the potentially large network sizes that can be produced by selecting a large number of rings and large number of neurons per population stage in

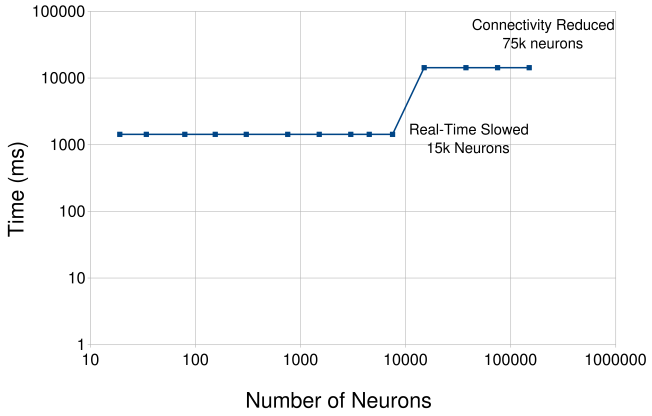


Fig. 6: Run time for the network sizes given in table I. For sizes less than 15004 neurons ($<$ 1000 neurons/layer), the simulation ran in real time, completing in 1430 ms. For widths of 1000 and 2500 neurons/layer, simulation time had to be slowed by a factor of 10 for the network to function, finishing in 14300 ms. For widths of 5000 and 10000, we slowed simulation time and reduced connection probability to 0.001 (so that each neuron has a small fan-in of 5-10 connections).

a ring, make it an excellent test of the hardware capabilities of the system. It should be kept in mind that where activity rates fall outside of the design intent of the system, we should not expect the simulation to succeed; if it does, the hardware and software is doing better than planned.

B. Functionality Tests

We ran the synfire ring model with 3 rings of depth 3, 5, and 7 layers respectively, using various widths as shown below in table I. Given that the ring depths are the same, the output neuron should produce a spike at the same time for each network (i.e. the network size should be immaterial). We ran the network until it either 1) produced an output spike; 2) reached the end of simulation without producing a spike; or 3) terminated abnormally (e.g. because the timer tick could not be serviced in time). For conditions 2) and 3) we attempted to modify network parameters until it did succeed, first trying to alter the mapping to pack more threads per core and fewer neurons per thread, next slowing the timer tick (so that simulation time was a fraction of real time), finally reducing the connection probability in the ring. Results are in fig. 6.

Neurons/layer	Total Neurons	Neurons/layer	Total Neurons
1	19	200	3004
2	34	300	4504
5	79	500	7504
10	154	1000	15004
20	304	2500	37504
50	754	5000	75004
100	1504	10000	150004

TABLE I: Synfire network sizes

As can be seen, for small to medium scales, real time could be preserved, at least with an astute choice of mapping; at the largest real-time scale (7504 neurons) the network succeeded with 2 neurons/thread and 16 threads/core - the largest network we could model at this degree of device sparseness on the available hardware: 1 box, 384 cores. At larger scales it was still possible to run the network successfully by slowing real time, and at the very largest scales, 150004 neurons maximum, we were able to get the model to succeed by severely constraining the connectivity, packing 32 neurons/thread on 16 threads/core. The evidence strongly indicates, however, that larger networks at any desired timescale factor can be achieved by increasing the size of the hardware system and packing fewer neurons per thread. 32 neurons/thread appears to be the maximum supportable limit; we tried various configurations with 64, 128, and 256 neurons per core, and all failed.

C. Performance Tests

We implemented a random recurrent neural network with sizes from 50 to 500,000 nodes and placed on one box. 80 percent of neurons were excitatory (having only excitatory target synapses) while 20 percent were inhibitory neurons. We tested both Izkikevich and LIF neural models and found that total simulation time was largely independent of both the model and number of synapses per neuron from 100-1000 synapses/neuron. Maximum run time was 20 s for the largest networks (Fig. 7).

As a reference for comparison with other systems, we looked up the best reported results in terms of synaptic events/core/s for SpiNNaker [29] and TrueNorth [30] and compared them with the best POETS result as seen in Table II.

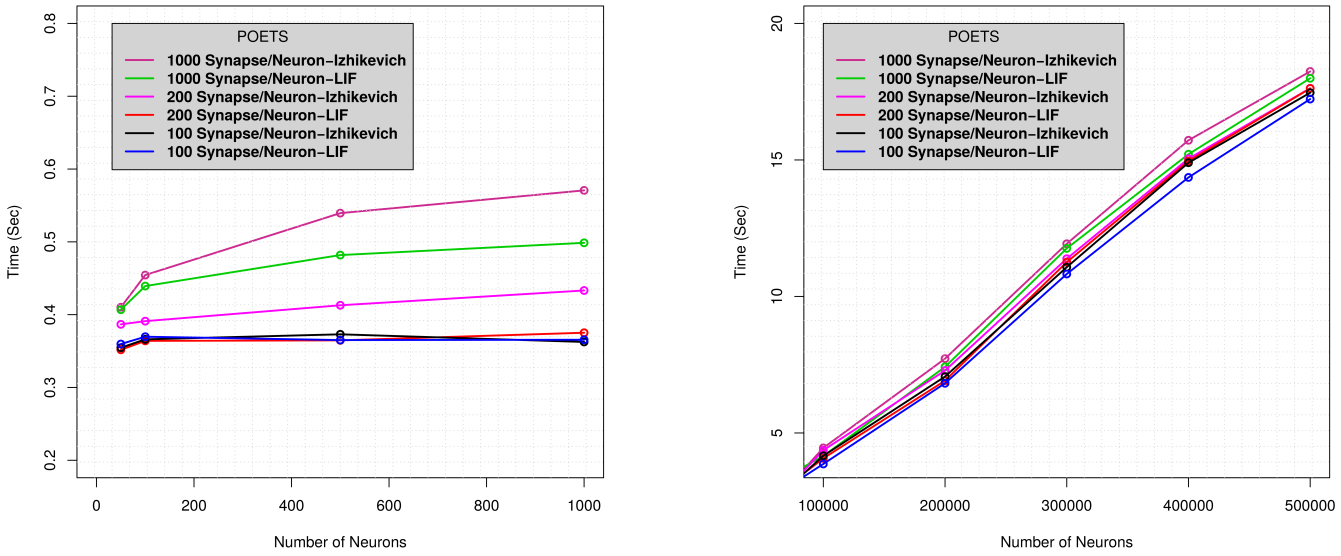
System	events/core/s
SpiNNaker	7.69×10^6
TrueNorth	14.2×10^6
POETS	16.0×10^6

TABLE II: Maximum event rate for neuromorphic systems

These numbers should be treated with caution. Figures in each case were obtained from different simulations, mapped in different ways with different neural model implementations, and reported differently for each platform. What constitutes a ‘core’ in each system is different and may not be exactly comparable. Therefore, these results should be considered at best indicative rather than conclusive. However, it does demonstrate that all 3 platforms have approximately the same order-of-magnitude peak performance.

V. Implications

In simulation, the number of input events per neuron per time step clearly dominates the time fidelity (and simulation size). Run time from 50 to 7500 neurons was almost the same, as the nodes could be assigned to different threads on each core simultaneously. Efficient communication between thread-local memory and CPU allows performance to be largely independent of the size of the synapic block from 1-1000



(a) Small networks

(b) Large networks

Fig. 7: Total runtime for LIF and Izhikevich neural models on POETS for various network sizes and connectivities. Run time includes the time to build and compile the model, and place it on the hardware, in addition to the simulation time proper.

synapses/neuron. However, time significantly increased as the event rate went up, and this will be magnified with STDP enabled, still more with higher-fidelity plasticity rules. We further note that it is worst-case, not average-case, event rate that matters. If at any point, the number of inputs to service take longer than the time step interval to process, the simulation will lose real-time fidelity, unless the time step is increased (sacrificing either simulation accuracy or speed). Simulations where threads could stay below a maximum message rate of 1000 messages/thread/ms tended to succeed, simulations where this maximum was exceeded, even occasionally, typically failed. This has important placement implications; it makes sense to use a connection-oriented placement which keeps the number of inputs per thread (or per core) approximately constant and below some maximum limit. The synfire ring is a somewhat contrived example that exposes this limit particularly vividly, but in more realistic networks one might expect that a similar placement strategy will be effective. This is likely to apply to almost any neuromorphic device which does not have fixed connectivity, and results in an elegant and simple general placement rule that can minimise *total* simulation time in systems where it must be remembered that the cost of simulation includes setup and configuration, not just run time.

The strength of an overlay platform like POETS is that it is able to reveal such behaviour early and inform design decisions, whether for a custom neuromorphic IC or for futher iterations of the overlay. Given the results seen, for example, it makes sense for future neuromorphic designs to devote resources to optimise spike-receive efficiency [31]. A future neuromorphic chip could, for example, benefit by having dedicated hardware or threads to run the spike-receive process whilst other hardware concurrently deals with periodic (or continuous-time) state updates [29]. This is an area for future exploration in POETS, where we are working on dedicated 'hard cores' to offload time-critical processing from the Tinsel general-purpose cores. Currently, sending was also constrained by the point-to-point send protocol, requiring a separate message for each target in a neuron's fanout; we have recently implemented a multicast routing capability which will relieve this bottleneck by allowing a single packet to reach all targets, like the SpiNNaker chip. We are also working on a general-purpose PyNN [32] front-end for POETS that can generate the XML overlay for arbitrary spiking networks as per the PyNN specification. Finally, in the current work, we have not yet explored STDP or learning; we are actively working on STDP implementations for learning spiking networks.

VI. Conclusions

What is the 'right' architecture for a neuromorphic chip, and how can one gain confidence that it will work as expected? While it is not clear that there is any one *ideal* architecture, a system like POETS allows different choices to be explored and debugged while still maintaining hardware speed. Many behaviours, whether intentional or faulty, do not emerge until real networks are tried running at real-time speeds, and we have identified with POETS at least one behaviour: packet-servicing capacity, that may be crucial for successful operation and which could explain previously-unclear limitations of existing neuromorphic platforms. This moves towards an

effective hardware prototyping model for new neuromorphic designs, where expensive custom silicon can be de-risked at the design exploration stage and chips fabricated with reasonable confidence in the validity of the design decisions. POETS may not be, strictly, neuromorphic hardware, but it is 'close enough' in its architecture to be an effective proxy for large-scale neuromorphics.

ACKNOWLEDGMENT

This work has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under EPSRC grant EP/N031768/1. We have also received support from industrial partners ARM, Ltd., the Culham Centre for Fusion Energy, the Chinese University of Medicine Hong Kong, e-Therapeutics Ltd., Imagination Technologies, Maxeler Technologies, NAG, and NMI. We gratefully acknowledge the support of these institutions.

REFERENCES

[1] M. A. C. Maher, S. P. Deweerth, M. A. Mahowald, and C. A. Mead, "Implementing neural architectures using analog VLSI circuits," *IEEE T. Circuits Syst.*, vol. 36, no. 5, pp. 643–652, May 1989.

[2] R. J. Vogelstein, U. Mallik, J. T. Vogelstein, and G. Cauwenberghs, "Dynamically Reconfigurable Silicon Array of Spiking Neurons With Conductance-Based Synapses," *IEEE T. Neural Networ.*, vol. 18, no. 1, pp. 253–265, Jan. 2007.

[3] S. Renaud, J. Tomas, Y. Bornat, A. Daouzli, and S. Saïghi, "Neuromimetic ICs With Analog Cores: An Alternative for Simulating Spiking Neural Networks," in *Proc. 2007 IEEE Int. Symp. Circuit. Syst. (ISCAS2007)*, 2007, pp. 3355–3358.

[4] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling," in *Proc. 2010 IEEE Int. Symp. Circuit. Syst. (ISCAS2010)*, 2010, pp. 1947–1950.

[5] J. Schemmel, L. Kriener, P. Müller, and K. Meier, "An Accelerated Analog Neuromorphic Hardware System Emulating NMDA- and Calcium-Based Non-Linear Dendrites," in *Proc. 2017 IEEE Int. Joint Conf. Neural Netw. (IJCNN 2017)*. IEEE Press, 2017, pp. 2217–2226.

[6] C. S. Thakur, R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "A Low Power Trainable Neuromorphic Integrated Circuit That Is Tolerant to Device Mismatch," *IEEE Trans. Cir. Syst. I: Regular Papers*, vol. 63, no. 2, pp. 211–221, Feb. 2016.

[7] J. Binas, D. Neil, G. Indiveri, S.-C. Liu, and M. Pfeiffer, "Precise deep neural network computation on imprecise low-power analog hardware," *arXiv:1606.07786*, Jun. 2016.

[8] R. Serrano-Gotarredona, et al., "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking," *IEEE T. Neural Networ.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[9] E. Neftci, E. Chicca, G. Indiveri, and R. Douglas, "A Systematic Method for Configuring VLSI Networks of Spiking Neurons," *Neural Comput.*, vol. 23, no. 10, pp. 2457–2497, Oct. 2011.

[10] Y. Hirai, "A 1000-Neuron System with One Million 7-bit Physical Interconnections," in *Advances in Neural Information Processing Systems 10: Proc. 10th Conf. Neural Information Processing Systems (NIPS1997)*. MIT Press, 1997, pp. 705–711.

[11] M. James and D. Hoang, "Design of Low-Cost, Real-Time Simulation Systems for Large Neural Networks," *J. Parallel and Distributed Computing*, vol. 14, no. 3, pp. 221–235, Mar. 1992.

[12] P. Ienne, T. Cornu, and G. Kuhn, "Special-Purpose Digital Hardare for Neural Networks: An Architectural Survey," *J. VLSI Signal Processing Systems*, vol. 13, no. 1, pp. 5–25, Aug. 1996.

[13] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "A 1.42TOPS/W Deep Convolutional Neural Network Recognition Processor for Intelligent IoE Systems," in *Proc. 2016 IEEE Int. Solid-State Cir. Conf. (ISSCC 2017)*. IEEE Press, 2016, pp. 264–266.

[14] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verheist, "ENVISION: A 0.26-to-10TOPS/W Subword-Parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI," in *Proc. 2017 IEEE Int. Solid-State Cir. Conf. (ISSCC 2017)*. IEEE Press, 2017, pp. 246–248.

[15] S. Pande, F. Morgan, S. Cawley, T. Bruintjes, G. Smit, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "Modular Neural Tile Architecture for Compact Embedded Hardware Spiking Neural Network," *Neural Process. Lett.*, vol. 38, no. 2, pp. 131–153, Jan. 2013.

[16] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. Furber, "SpiNNaker: A 1W 18-core System-on-Chip for Massively-Parallel Neural Network Simulation," *IEEE J. Solid-St. Circ.*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.

[17] P. A. Merolla, et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[18] C. Eliasmith, "A unified approach to building and controlling spiking attractor networks," *Neural Comput.*, vol. 17, no. 6, pp. 1276–1314, Jun. 2005.

[19] A. Neckar, S. Fok, B. V. Benjamin, T. Stewart, N. N. Oza, A. R. Voelcker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model," *Proc. IEEE*, vol. 107, no. 1, pp. 144–164, Jan. 2019.

[20] M. S. Zaveri and D. Hammerstrom, "Performance/price estimates for cortex-scale hardware: A design space exploration," *Neural Networks*, vol. 24, no. 3, pp. 291–304, Apr. 2011.

[21] A. M. Zyarah, N. Soures, L. Hays, R. B. Jacobs-Gedrim, S. Agarwal, M. Marinella, and D. Kudithipudi, "Ziksa: On-Chip Learning Accelerator with Memristor Crossbars for Multilevel Neural Networks," in *Proc. 2017 IEEE Int. Symp. Circuit. Syst. (ISCAS2017)*. IEEE Press, 2017.

[22] A. K. Fidjeland, E. B. Roesch, M. P. Shanahan, and W. Luk, "NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs," in *Proc. 20th IEEE Int. Conf. Application-Specific Syst. Arch. Process.*, 2009, pp. 137–144.

[23] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive – A Field-Programmable Custom Computing Machine for Extreme-Scale Real-Time Neural Network Simulation," in *Proc. 2012 IEEE Int. Symp. Field-Programmable Custom Comput. Machines*. IEEE Press, 2012, pp. 133–140.

[24] C. D. Schuman, A. Disney, and J. Reynolds, "Dynamic Adaptive Neural Network Arrays: A Neuromorphic Architecture," in *Proc. Machine Learning in High-Perf. Comput. Environments (MLHPC 2015)*. Association for Computing Machinery, 2015.

[25] A. Cassidy, S. Denham, P. Kanold, and A. Andreou, "FPGA Based Silicon Spiking Neural Array," in *Proc. 2007 IEEE Biomed. Circuit. Syst. Conf. (BIOCAS 2007)*. IEEE Press, 2007, pp. 75–78.

[26] M. Naylor, S. W. Moore, and D. Thomas, "Tinsel: a manythread overlay for FPGA clusters," in *Proc. 29th International Conf. on Field Programmable Logic & Applications (FPL 2019)*. IEEE Press, 2019.

[27] E. Izhikevich, "Simple Model of Spiking Neurons," *IEEE T. Neural Networ.*, vol. 14, pp. 1569–1572, Nov. 2003.

[28] A. D. Brown, J. E. Chad, R. Kamarudin, K. J. Dugan, and S. B. Furber, "SpiNNaker: Event-Based Simulation - Quantitative Behavior," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 3, pp. 450–462, Jul.-Sep. 2018.

[29] J. Knight and S. B. Furber, "Synapse-Centric Mapping of Cortical Models to the SpiNNaker Neuromorphic Architecture," *Front. Neurosci.*, Sep. 2016.

[30] F. Akopyan, et al., "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Programmable Neurosynaptic Chip," *IEEE Trans. Comput.-Aided Design Integrated Cir. Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[31] O. Rhodes, L. Peres, A. G. D. Rowley, A. Gait, L. A. Plana, C. Brenninkmeijer, and S. B. Furber, "Real-time cortical simulation on neuromorphic hardware," *Phil. Trans. Roy. Soc. A - Math., Phys. and Eng. Sci.*, vol. 378, Dec. 2019.

[32] A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "PyNN: a common interface for neuronal network simulators," *Front. Neuroinform.*, vol. 2, Jan. 2009.