INVITED PAPER

The SpiNNaker Project

This paper describes the design of a massively parallel computer that is suitable for computational neuroscience modeling of large-scale spiking neural networks in biological real time.

By Steve B. Furber, *Fellow IEEE*, Francesco Galluppi, Steve Temple, and Luis A. Plana, *Senior Member IEEE*

ABSTRACT | The spiking neural network architecture (SpiNNaker) project aims to deliver a massively parallel million-core computer whose interconnect architecture is inspired by the connectivity characteristics of the mammalian brain, and which is suited to the modeling of large-scale spiking neural networks in biological real time. Specifically, the interconnect allows the transmission of a very large number of very small data packets, each conveying explicitly the source, and implicitly the time, of a single neural action potential or "spike." In this paper, we review the current state of the project, which has already delivered systems with up to 2500 processors, and present the real-time event-driven programming model that supports flexible access to the resources of the machine and has enabled its use by a wide range of collaborators around the world.

KEYWORDS | Brain modeling; multicast algorithms; multiprocessor interconnection networks; neural network hardware; parallel programming

I. INTRODUCTION

THE spiking neural network architecture (SpiNNaker) project is motivated by the grand challenge of understanding how information is represented and processed in the brain [1]. Most of the frontiers of science are concerned with the very small, such as subatomic particles, or the very large, such as exploring the outer regions of the universe. Yet there remains a great unsolved scientific mystery at a very human scale: how does the brain, an organ that we could readily hold in our hands and observe with the naked eye, perform its role that is so central to all of our lives?

Manuscript received October 2, 2013; revised January 4, 2014; accepted February 2, 2014. Date of publication February 27, 2014; date of current version April 28, 2014. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under GrantEP/G015740/01.
The authors are with the School of Computer Science, University of Manchester, Manchester M13 9PL, U.K. (e-mail: steve.furber@manchester.ac.uk; francesco.galluppi@gmail.com; temples@cs.man.ac.uk; plana@cs.man.ac.uk).

Digital Object Identifier: 10.1109/JPROC.2014.2304638

"Wet" neuroscience has told us a great deal about the basic component—the neuron—from which the brain is constructed. Brain imaging tells us yet more about how activity moves around the brain as we perform certain mental functions. The former is concerned with individual neurons up to groups of tens or perhaps hundreds; the latter looks at the collective activity of many millions of neurons. But between these scales there are a few orders of magnitude of scale for which there exists no scientific instrument except the computer model, and it is at these intermediate scales, we suggest, that all the interesting information processing takes place.

Our conclusion is that, if we wish to fully understand how the brain represents and processes information, we need to build computer models to test hypotheses of how the brain works.

A. Neurons and Spikes

What sort of computer is required for such brain modeling to work?

The human brain is generally viewed as comprising somewhat under 100 billion neurons, where each neuron is a multiple-input–single-output device.

There is some debate about the role of the more numerous glial cells that form the structure upon which the neurons build the brain, and, in particular, the role of astrocyte cells in synaptic plasticity [2], so any general-purpose system should aim to accommodate these issues in case they prove to be important.

Neurons communicate principally through action potentials, or "spikes." These are simply asynchronous impulses where, as a result of the electrochemical regeneration process used to ensure the reliable propagation of these signals along long biological "wires," information is conveyed only in the identity of the neuron that spiked and the time at which it spiked. The height and the width of the impulse are largely invariant at the receiving synapse. This has led to the widespread adoption of the address event representation (AER) encoding of

neural activity [3], [4], where the information flow in a network is represented as a time series of neural identifiers.

There are some notable exceptions to the completeness of the AER view of information flow. Some neurons transport and emit neuromodulators, such as dopamine, that have a global effect on neurons within a neighborhood region; other neurons make direct contact through "gap" junctions that make an electrical connection from one neuron to its neighbor. However, in much of the brain, the primary real-time information flow is in the spikes that, in a model, are represented by AER. A general-purpose computer-modeling platform should offer mechanisms to support these other information flows while giving first-class support to AER "spikes."

B. Computer Models

What computer power and architecture are required to support a real-time model of the human brain?

The simplest estimate of an answer to this question suggests that there are around 10^{15} synapses in the brain, with inputs firing at an average rate of 10^{1} Hz, and each synaptic event requires perhaps 10^{2} instructions to update the state of the postsynaptic neuron and implement any synaptic plasticity algorithm. These figures lead to an estimate of 10^{18} operations per second, the performance of an exascale machine. Exascale high-performance computers do not yet exist, though recently the Chinese Tianhe 2 machine has achieved 3×10^{16} floating-point operations per second [5], so exascale computing is not too far away.

However, raw computer performance is not the only issue here. The communication patterns in the brain are based on sending very small "packets" of information through complex paths to many targets. High-performance computers, on the other hand, are generally optimized for point-to-point communication of large data packets. This mismatch leads to significant inefficiency in the mapping of brain-scale spiking neural networks onto conventional cluster machines and high-performance computers.

C. SpiNNaker

The SpiNNaker machine is a computer designed specifically to support the sorts of communication found in the brain. Recognizing the huge computational requirements of the task, SpiNNaker is based on massively parallel computation, and the architecture will accommodate up to a million microprocessor cores, the limit being defined by budget and architectural convenience rather than anything fundamental.

The key innovation in the SpiNNaker architecture is the communications infrastructure, which is optimized to carry very large numbers of very small packets, in contrast to the conventional cluster and high-performance computer communications system which, as noted above, are optimized for large data packets. Each packet carries a single neural "spike" event in a 40-b packet, 32 b of which are the AER identifier of the neuron that spiked and 8 b are management bits identifying the packet type, and such like. (The choice of a 32-b AER identifier is not a fundamental limitation of the architecture, and could be increased in a future implementation to accommodate larger neural models.) The time of the AER spike is implicit; the communications infrastructure can deliver a packet in much less than a millisecond, which is the requirement for real-time neural modeling.

Although SpiNNaker's design is centered on packet-switched support for AER "spikes," it can also support non-AER information flows through the same communication mechanism delivering discrete (typically 1 ms) updates to continuously variable parameters.

In order to achieve efficient massively parallel operation, SpiNNaker's design accepts certain compromises, one of which is the requirement for deterministic operation. The asynchronous nature of the communications system leads to nondeterministic ordering of packet reception, and occasionally packets may be dropped to avoid communication deadlock. It is possible to reimpose deterministic operation and lockstep operation to match a conventional sequential model under certain conditions, but this is not the natural or most efficient way to operate the machine.

D. Paper Organization

This paper is a review of the SpiNNaker project and a tutorial on the use of the machine. The contributions and structure of the paper are as follows.

- We present an overview of the architecture (Section II) and of the hardware implementation (Section III).
- We present the system software (Section IV), describe the event-driven software model (Section V), the API that supports this (Section VI), and a simple example program that runs on top of the API (Section VII).
- We present the partitioning and configuration manager (PACMAN, Section VIII) that conceals the physical structure of the machine.
- Finally, we describe some typical applications that run on the machine (Section IX), our future plans for larger scale machines (Section X), discuss related work (Section XI), and draw our conclusions (Section XII) from our experience with the machine at this stage in its development.

II. ARCHITECTURE OVERVIEW

A detailed description of the architecture of the machine has been presented earlier [6], so here we present the key features of the architecture that are germane to what follows (see Fig. 1).

A SpiNNaker machine is a homogeneous 2-D multiple instruction, multiple data array of processing nodes where each node incorporates 18 ARM968 processor cores each with 96 kB of local memory, 128 MB of shared memory, a

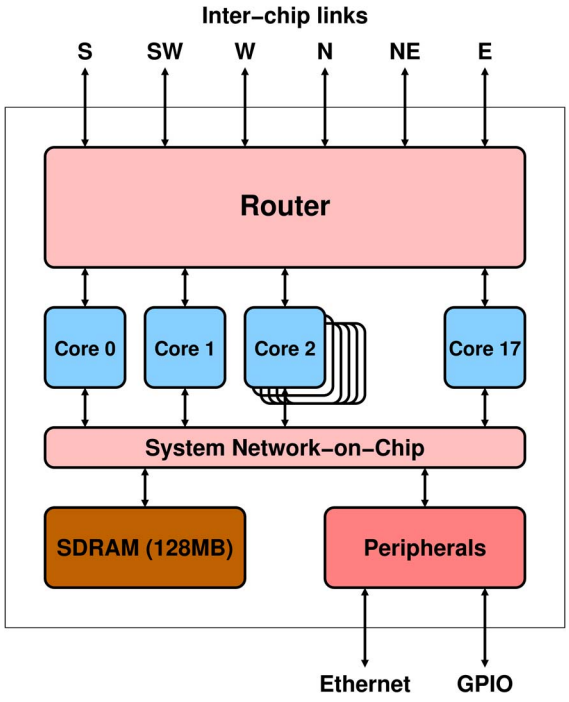

Fig. 1. ***Principal architectural components of a SpiNNaker node.***

packet router, and general system support peripherals. Each processor core is a general-purpose 200-MHz 32-b integer processor with no floating-point hardware, so arithmetic is generally implemented as fixed point.

A. Achievable Performance

Each core can model a few hundred point neuron models, such as leaky integrate and fire or Izhikevich's model, with the order of 1000 input synapses to each neuron. In practice, a number of different constraints may limit the number of neurons a processor core can support in real time, but often the compute budget is dominated by input connections—an incoming spike passing through an individual synapse—which imposes an upper limit on the (number of neurons) × (number of inputs per neuron) × (mean input firing rate). In principle, a processor core can support up to 10 million connections/s, though the current software implementation saturates at about half this throughput, and plastic synapse models reduce it considerably further.

B. Spikes and Packets

The key innovation in the SpiNNaker architecture is a lightweight multicast packet-routing mechanism that supports the very high connectivity found in biological brains. The mechanism is an extension of conventional AER [3], [4]. When the software running on a processor identifies that a neuron should emit a spike, it simply issues a packet that identifies the spiking neuron. The issuing processor has no idea of where that packet will be conveyed to—that is entirely the responsibility of the routing fabric.

Each node incorporates a packet router that inspects each packet to look at its source, and routes it accordingly to any subset of its 18 local processors and/or any subset of its six neighbor nodes using multicast transmission (which has been shown to be optimal for neural applications [7]) in a 2-D triangular mesh. The selected routes are determined by tables in the router that are initialized when the application is loaded into the machine.

As the packet source identifier is 32 b, it is infeasible to implement full routing tables for every possible source, so a number of optimizations are employed to keep the table sizes reasonable.

- The tables are implemented using content addressable memory (CAM), and entries are required only for those packets that pass through a node.
- The CAM uses four states: match 0, match 1, match all, and no match. This allows a single CAM entry to route all of those neurons in a population with common routing requirements.
- Where no CAM entry matches a source identifier, a default routing mechanism allows the packet to pass straight through the node.

These optimizations allow a routing table with 1024 entries to be sufficient at each node. We will return to the matter of initializing these tables in Section VIII.

C. Processor Disposition

Each SpiNNaker node selects one of its 18 processor cores to act as "monitor processor." This selection is flexible for fault-tolerance reasons. Once selected, the monitor is assigned an operating system support role. Sixteen of the remaining processors are assigned application support roles, and the 18th processor is held in reserve as a fault-tolerance spare, though on a proportion of nodes, the 18th processor may be faulty as nodes with only 17 functional processors are accepted in production to enhance yield.

III. CHIPS, PACKAGES, BOARDS, AND SYSTEMS

The physical implementation of the SpiNNaker architecture has also been described in detail elsewhere [8], so again we will restrict ourselves here to the relevant highlights.

A. Chips and Packages

Each SpiNNaker node is implemented in a single 19-mm square 300 ball grid array package. The package houses a custom-designed multiprocessor system-on-chip integrated circuit that includes the 18 ARM968 processors, each with its local 32-kB instruction memory and 64-kB data memory, interconnected through a self-timed

network on chip to various on-chip shared resources and a second chip, a 128-MB low-power mobile dual-data-rate (DDR) SDRAM. The two chips are stacked onto the package substrate and interconnected using gold wire bonding (Fig. 3). The aggregate SDRAM bandwidth has been measured to be 900 MB/s [8].

B. Boards

The packages are then assembled onto printed circuit boards (PCBs; see Fig. 2). The chip-to-chip connections on the PCB are direct wired connections using a self-timed 2-of-7 non-return-to-zero protocol to transmit 4-b symbols with two wire transitions, plus one wire transition for the acknowledge response.

In principle, these direct connections could be used to build a SpiNNaker machine of arbitrary size, but for practical reasons the machine is constructed from 48-node PCBs, and the PCB-to-PCB connections use high-speed serial links where eight chip-to-chip links are multiplexed through each serial link using Xilinx Spartan6 field-programmable gate arrays (FPGAs).

C. Systems

SpiNNaker systems of varying sizes can then be assembled from one or more of the 48-node PCBs. There is also a smaller four-node board that is very convenient for training, development, and mobile robotics. The largest machine, incorporating over a million ARM processor cores, will comprise 1200 48-node boards in ten machine room cabinets and will require up to 75 kW of electrical power (peak).

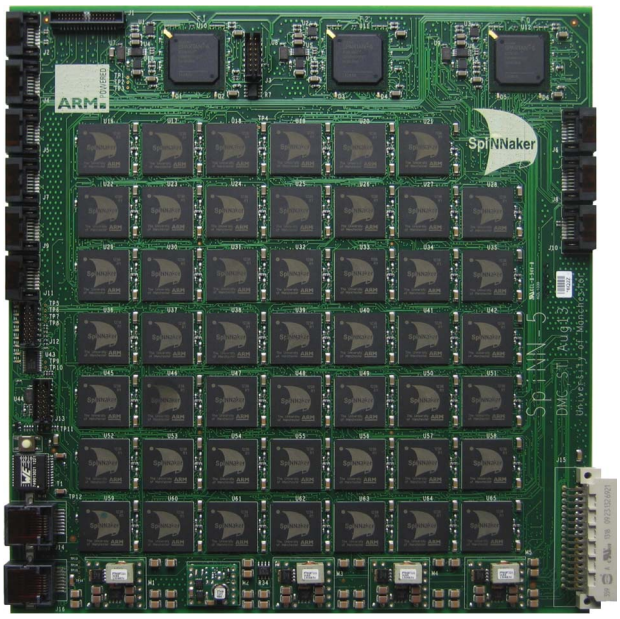

Fig. 2. *A 48-node SpiNNaker PCB. This circuit board incorporates 48 SpiNNaker packages (center) with a total of 864 ARM968 processor cores, three FPGAs (top) for high-speed inter-PCB communications through serial advanced technology attachment connectors (top left and right), with onboard power regulation (bottom).*

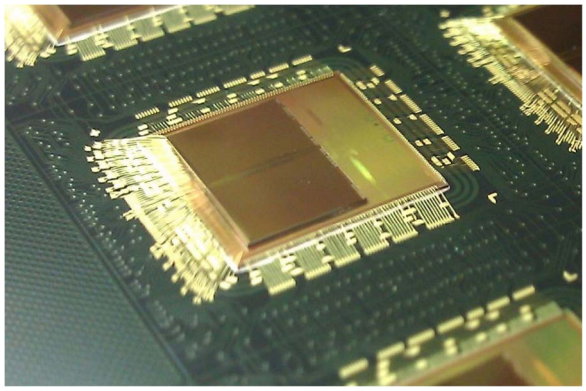

Fig. 3. *Inside a SpiNNaker package. The SpiNNaker chip is mounted on the substrate, then a 128-MB mobile DDR SDRAM is stacked on top of it, and the connections are made inside the package with gold wire bonding. The packaging was carried out by Unisem Europe Ltd.*

IV. SPINNAKER SYSTEM SOFTWARE

SpiNNaker software can be categorized into that which runs on the SpiNNaker system itself and that which runs on other systems, some of which may interact with SpiNNaker. The majority of software that runs on the SpiNNaker chips is written in C. This software can be subdivided into control software (a primitive operating system) and application software which performs the user's computations.

The primary interface between SpiNNaker systems and the outside world is Ethernet and IP-based protocols. Every SpiNNaker chip has an Ethernet interface and typically one chip per PCB uses this interface. This is used to download code and data to SpiNNaker and to gather results from applications. For some applications, this (100 Mb/s) interface is a bottleneck on getting data to and from SpiNNaker, and we are investigating the use of gigabit links provided by FPGAs on SpiNNaker PCBs to improve this.

A. SpiNNaker Software

The control software that runs on SpiNNaker systems is known as the SpiNNaker Control and Monitor Program (SC&MP). The SpiNNaker chips contain primary bootstrap code which allows the loading of code via the Ethernet interface or the interchip links, and this is used to load SC&MP, initially via an Ethernet interface to a single chip. SC&MP is then propagated to the entire system over the interchip links; it runs continuously on the core that has been selected as the monitor processor and provides a range of services to the outside world to allow applications to be loaded on the remaining 16 or 17 application cores on each chip.

A simple packet protocol known as SpiNNaker Datagram Protocol (SDP) is used within the SpiNNaker system. SC&MP acts as a router for SDP packets allowing them to be sent to or from any core in the system and also via Ethernet to external endpoints. This protocol forms the basis for application loading and high-level communication between SpiNNaker chips and/or external machines. Within individual chips, SDP packets are exchanged between cores using a shared-memory interface. Between chips, SDP is transported as sequences of point-to-point packets conveyed by the interchip links. To carry SDP out of the system, the packets are embedded in UDP/IP packets and sent via the Ethernet interface to external endpoints.

A SpiNNaker "application" is a program that runs on one or many of the application cores on a SpiNNaker system. It will typically be written in C and utilize either SDP or multicast packets for its communication needs. Because of the limited code and data size provided by the on-chip memories in SpiNNaker, there is little room for operating system support and so only minimal ancillary code can be loaded along with the application. Each application is linked with a support library known as SpiNNaker Application Runtime Kernel (SARK). SARK provides startup code for the application core to set up the runtime environment for the application. It also provides a library of functions for the application such as memory allocation and interrupt control. SARK also maintains a communications interface with SC&MP running on the monitor processor that allows the application to communicate with and be controlled by other SpiNNaker chips or external systems. Protocols running on top of SDP are used to achieve this functionality.

An application is built using an ARM cross compiler and linked with SARK and any other runtime libraries that it requires. The output file from the linking stage is converted to a format known as application load and execute (APLX) which is understood by a simple loader which is part of SC&MP. The APLX file can then be downloaded to the SpiNNaker system where it is loaded into the appropriate parts of memory of the relevant application cores by the SC&MP loader.

Most SpiNNaker applications make use of an event management library known as the Spin1 API. This provides facilities for associating common interrupts with event handling code and for managing queues of events. While the processor is not processing events it is in a low-power sleep mode. This API can be viewed as a software layer between the user's application and the underlying hardware. To facilitate SpiNNaker program development using the API, an emulator has been developed which provides the same set of library calls as the Spin1 API but which runs on a Linux workstation. This allows users without SpiNNaker hardware to develop and debug SpiNNaker applications and to familiarize themselves with the programming model.

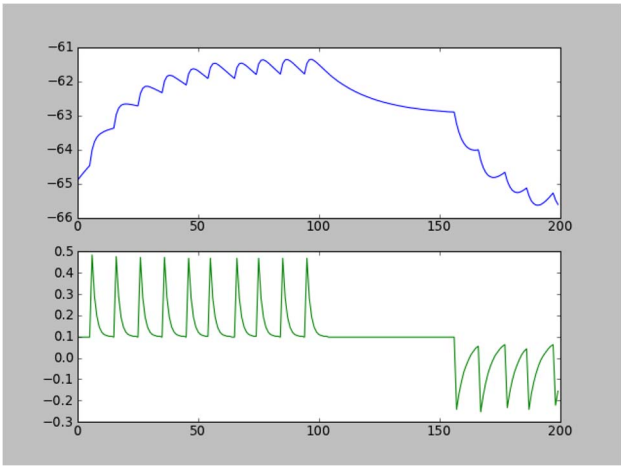

Fig. 4. ***Example output from a SpiNNaker visualizer.***

B. Host Software

We refer to the workstation that controls a SpiNNaker system as the "host." A variety of SpiNNaker-related host software has been developed within the project. A number of tools have been developed to download applications to SpiNNaker systems. The "ybug" program provides a command line interface for this function and also allows scripted control of the system. A number of application programmer interfaces (APIs) that implement interfaces based on SDP have been developed in C, Perl, and Python. These allow programmed control of a SpiNNaker system to allow applications to be downloaded and controlled and results uploaded.

In addition, a number of "visualizer" applications have been produced which allow the results of SpiNNaker applications to be viewed on the host system. The simplest of these just allows plain text output to be displayed on the host while more sophisticated visualizers [9] display data in graphical form such as the raster plot of firing spikes in a neural network simulation or the potentials inside a single neuron (Fig. 4).

The provision of input data to SpiNNaker applications can also require host software to provide these data. One such application is the "spike server" which is used to provide spikes (neural events) in real time to a neural simulation running on SpiNNaker.

A significant part of the SpiNNaker software effort has been the development of programs that map complex problems onto the SpiNNaker hardware. A typical example is a neural network simulation where individual neurons or groups of neurons have to be allocated to cores in the system and the routing tables set up to allow them to communicate appropriately for the connectivity of the network. The "PACMAN" program, which is described in Section VIII, is typical of this class of program [10].

Fig. 5 shows the arrangement of the various software components which make up a SpiNNaker system.

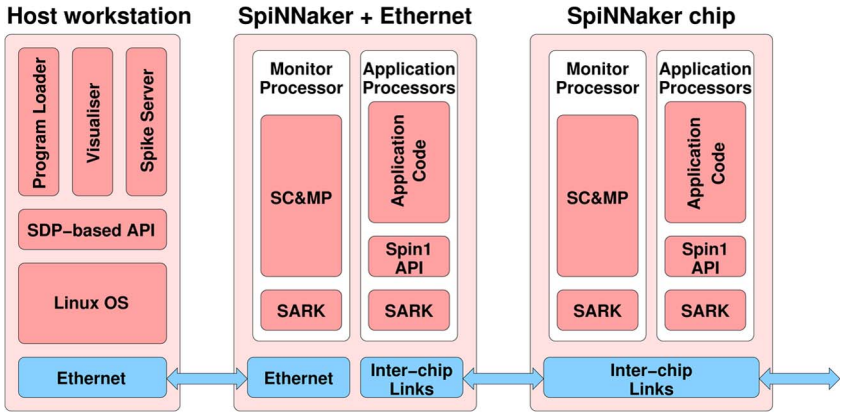

Fig. 5. ***The various software components running on the host machine, the root node, and other SpiNNaker nodes.***

V. EVENT-DRIVEN SOFTWARE MODEL

The programming model employed on SpiNNaker is that of a real-time event-driven system. The application processors have a base state, which is halted and waiting for an interrupt, contributing to the overall energy efficiency of the system. In the standard neural modeling application, there are three principal events that cause the processor to wake up.

1) An incoming spike packet. This will usually cause the processor to initiate a direct memory access (DMA) transfer from SDRAM of the synaptic data structures associated with the source of this spike.
2) DMA complete. Once the synaptic data have been transferred, the processor must process the data.
3) One-millisecond timer tick. Each processor has a local timer that marks the passage of time, and each millisecond (typically, the interval is programmable) the processor will compute a further integration step in the neuron dynamics.

Of course, these events are asynchronous and unpredictable, so the software running on the processor must be capable of prioritizing the events and handling multiple overlapping requests. This is achieved through the use of a real-time kernel that underpins the event-driven operation of each application processor, and presents a straightforward API to the user, who can build applications on top of the API entirely in C.

VI. SPINNAKER APPLICATION PROGRAMMING INTERFACE

The SpiNNaker application programming interface (spin1 API) [11] provides an execution environment that supports a lightweight, event-driven programming model. A central goal of the model is to save energy by keeping the cores in a low-power state, only responding to events of interest. To this effect, application programs do not control execution flow; they can only indicate the functions, referred to as callbacks, to be executed when specific events, such as the arrival of a packet, the completion of a DMA transfer, or the lapse of a periodic time interval, occur. The callback mechanism is also used to hide the details of the interrupt subsystem, which is handled directly and efficiently by the API.

Fig. 6 shows the basic architecture of the event-driven framework. Application developers write callback routines that are associated with events of interest and register them with the API at a priority level, which defines them as queueable or non-queueable. When the corresponding event occurs, the scheduler either executes the callback immediately and atomically (in the case of a non-queueable callback) or places it into a scheduling queue at a position according to its priority (in the case of a queueable callback). When control is returned to the dispatcher (following the completion of a callback) the highest priority queueable callback is executed. Queueable callbacks do not necessarily execute atomically: they may be preempted by non-queueable callbacks if a corresponding event occurs during their execution.

The dispatcher goes to sleep (in the low-power consumption "wait for interrupt" state, where the processor core clock is turned off) when the callback queues are empty and will be awakened by any event. Application developers can designate one non-queueable callback as the preeminent callback, which has the highest priority and can preempt other non-queueable callbacks as well as all queueable ones. The API provides support for callbacks to control entry and exit from critical sections to prevent higher priority callbacks interrupting them at a bad time, e.g., during access to a shared resource.

This real-time kernel is scalable to very large numbers of processors, but is best suited to relatively simple models running on each processor. Clearly, the system will come to a halt if no events are generated, and real-time performance will be lost if a processor is overwhelmed by incoming events. In practice, careful mapping of a model onto the system can avoid both eventualities.

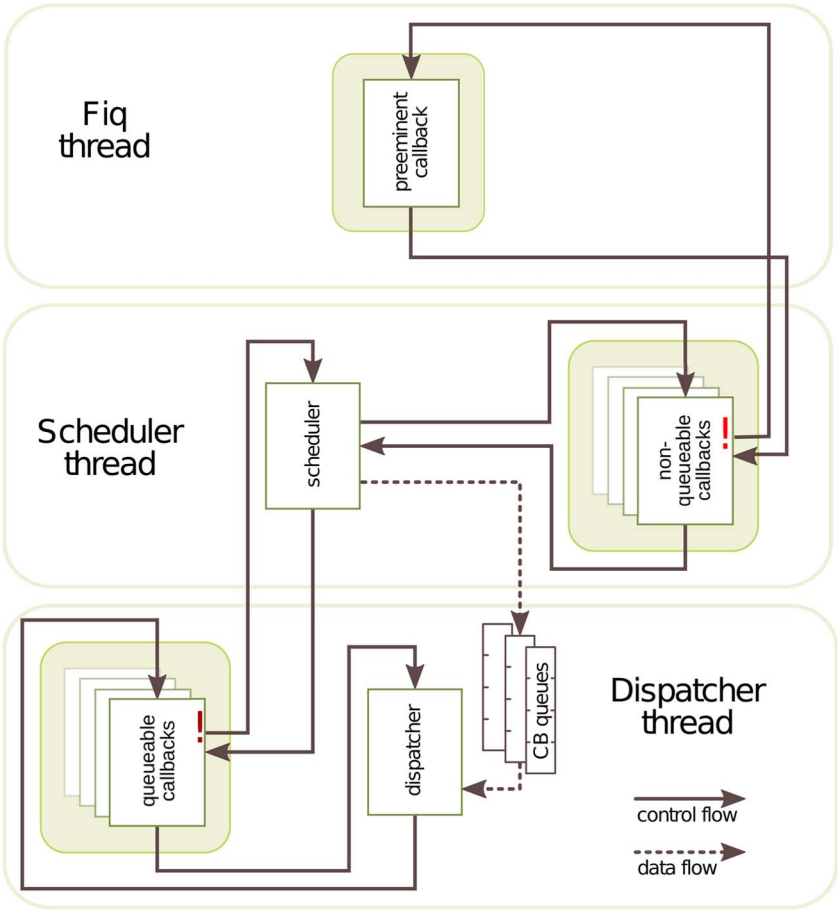

Fig. 6. ***Event-driven software framework.***

VII. EXAMPLE LOW-LEVEL APPLICATION

As a simple example of a parallel program that runs on top of the SpiNNaker API, here are the key features of a simple example that implements Conway's Life cellular automaton.

First, the program should include the API calls:

```
#include ⟨spin1_api.h⟩
```

Then, we need routines to set up the initial state of the automaton and the routing tables. In this case, setting up the routing tables is by far the most complex aspect of the programming task as the Life neighbor connections must be established between processors across chip boundaries.

```
void set_up_route_tables
(uintchip, uintcore){...}
void init_Life_state (uintchip, uintcore){...}
```

Now we must define the event-driven callback routines. In this example, the relevant events are timer tick and an incoming packet:

```
void tick_callback (uintticks, uintdummy){...}
void pkt_in (uintkey, uintdata){...}
```

The simulation is started on each processor from `c_main`. The chip and core addresses are found, then the initialization routines are called:

```
void c_main (void)
{
    uint chip = spin1_get_chip_id ();
    uint core = spin1_get_core_id ();
    set_up_route_tables (chip, core);
    init_Life_state (chip, core);
```

The timer period is set to 1 ms, and the event callbacks are set up with appropriate priorities (packet received is usually at the highest priority):

```
spin1_set_timer_tick(1000);
spin1_callback_on (TIMER_TICK,
tick_callback, 1);
spin1_callback_on(MC_PACKET_RECEIVED,
pkt_in, −1);
```

Finally, the simulation is started:

```
    spin1_start();
}
```

VIII. PARTITIONING AND CONFIGURATION MANAGER

The example described in Section VII shows how API-based applications can set up the simulation parameters, SDRAM content, and routing tables with an algorithmic process. While for simple or highly structured problems this is possible, modeling networks with arbitrary interconnectivity and arbitrary neural types is a problem where a further level of abstraction can be introduced. Configuring a million-core machine, with each core modeling up to a thousand neurons and a million synapses, rapidly becomes an intractable problem: one billion neurons need to be mapped and one trillion synapses need to be routed to implement a user-specified model.

To solve this problem, we introduce PACMAN [10], a software layer that enables users to write their model using a standardized interface, translate it, and run it on SpiNNaker. The software is designed to keep different concerns separated: users interface with the platform through domain-specific, neural languages already present in the scientific milieu, such as PyNN [12] or Nengo [13]. PACMAN is the set of algorithms that translate a model into machine-executable code. Such algorithms operate on data representing the network model, information about the system (topology, fault status, etc.), and methods for data structure translation.

PACMAN maps, routes, and translates network models using populations of neurons and projections between them, rather than single neurons and synapses. This approach reduces the complexity of the algorithms involved in the translation process, by exploiting the hierarchies present in a neural network. This choice is justified by studies on the structure of the central nervous systems, where functionally segregated areas are interconnected by axonal pathways [14], and where cortical areas show a remarkably regular laminar structure, with different layers of neurons stereotypically connected in a canonical circuit [15]. Finally, many neural languages [12], [13], [16], [17] use this abstraction natively, making it a natural choice.

Using a neural language as a user interface makes the platform more accessible to nonexperts, giving the users a familiar environment to develop models and analyze results, while hiding the complexity of configuring a parallel system and encouraging model sharing across different platforms. The translation process is performed by PACMAN as illustrated in Fig. 7, which shows the flow of the algorithms used to translate and execute the models (left) and the data representations they work on (right).

The model is represented in terms of populations and projections in the model view. It is then partitioned, splitting populations, while preserving their interconnectivity structure, accordingly to machine-specific constraints, depending on the neural and synaptic capacity of each core. The model is represented in a digraph-like structure (PACMAN view), and then mapped and routed on a physical machine instance (system view), using the information present in the system library. Finally, the whole model is translated into machine-executable code for each component (ARM cores, SDRAM, routers), using the translation mechanism stored in the model library, loaded onto the system, and executed.

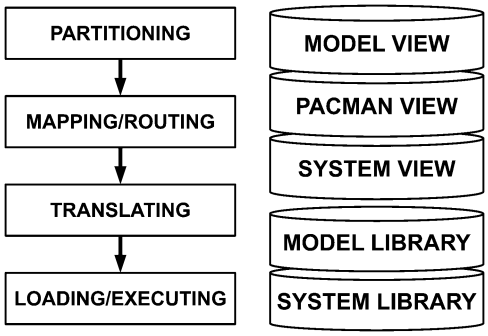

Fig. 7. *The flow of algorithms (left) and the data representations they work on (right) within PACMAN.*

A simple example network is illustrated in Fig. 8 (left): excitatory and inhibitory populations are recurrently interconnected. The ratio of excitatory to inhibitory neurons is set to 4 : 1 to keep a balance between excitation and inhibition.

The network can be represented in PyNN [12], first by creating the two populations of neurons, for a total of n neurons, with a set of parameters:

```
cell_params = {'tau_refrac': 5.0, 'v_thresh':−50.0,
 'v_reset':−60.0, 'tau_m': 20.0, 'tau_syn_E': 5.0,
 'tau_syn_I': 10.0, 'v_rest':−49.0, 'cm': 0.2}
ex = Population(n − n/5, IF_curr_exp,
cell_params)
in = Population(n/5, IF_curr_exp, cell_params)
```

The resting potential is located above the threshold potential to induce spontaneous firing in all cells. Populations are interconnected by means of a FixedProbability-Connector, which connects all the neurons in the presynaptic population

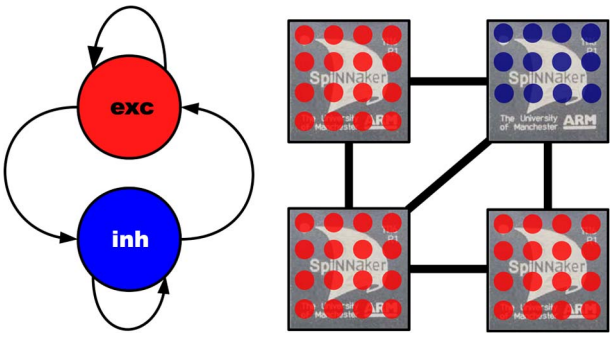

Fig. 8. *Example network (left) with one excitatory and one inhibitory population with a size ratio of 4 : 1 is mapped by PACMAN onto 60 processors on four SpiNNaker chips (right).*

to all the neurons in the postsynaptic population with a probability p, weight w, and delay d.

```
con = FixedProbabilityConnector (p_connect =
      p, w, d)
e_e = Projection (ex, ex, con, target =
      'excitatory')
e_i = Projection (ex, in, con, target =
      'excitatory')
i_i = Projection (in, in, con, target =
      'inhibitory')
i_e = Projection (in, ex, con, target =
      'inhibitory')
```

All the projections coming from the excitatory population target excitatory synapses; conversely, all the projections coming from the inhibitory population target inhibitory synapses.

PACMAN automatically partitions and maps the network as illustrated in Fig. 8 (right), which shows an example where the total number of neurons n is 6000, and each core maps 100 neurons. As a result, the model needs to be partitioned into 48 excitatory and 12 inhibitory subgroups, each to be allocated to a single core of a physical machine, with the system library providing the geometry (in this case, a four-chip board) and the functional status of the platform. The model library provides the translation methods for the IF_curr_exp neuron type (a leaky integrate and fire with exponential decaying synapses), its parameters, and its synapses. Fig. 9 shows results of 1 s of simulation in the form of a raster plot, where each dot represents a spike from a neuron (ordinate) in time (abscissa). Red (blue) dots represent spikes from excitatory (inhibitory) neurons; the interconnectivity parameters are set to give rise to the oscillatory activity shown in the figure.

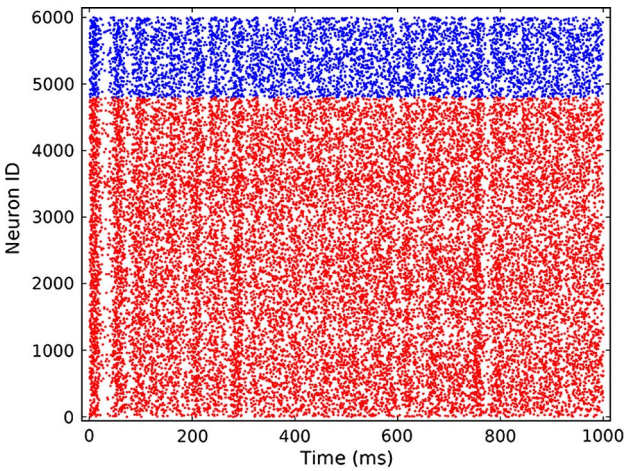

Fig. 9. *Raster plot of the results of running the simulation of the network shown in Fig. 8. Each dot represents one neural spike; red dots are excitatory neurons, and blue dots are inhibitory neurons. Oscillatory activity is visible across the network.*

IX. TYPICAL APPLICATIONS

In this section, we review some scenarios highlighting the flexibility of the SpiNNaker platform, and present an experiment running on a robot equipped with AER sensors and a 48-node SpiNNaker board.

With the hardware and software infrastructure presented in the previous sections we have simulated networks with up to 250 000 neurons and 80 million synapses in real time on a 48-node SpiNNaker board (as shown in Fig. 2) within a power budget of 1 W per SpiNNaker package (containing a SpiNNaker chip and a 128-MB SDRAM; see Fig. 3). In terms of spike delivery (the dominant cost in neural simulations [18]) and power consumption, these experiments show 1.8 billion connections per second, using a few nanojoules per event and per neuron [19], and represent the maximum sustainable throughput of the system with the current software infrastructure.

Good power efficiency has also been demonstrated in a biologically plausible model of cortical microcircuitry inspired by previous work [15], [20], comprising 10 000 Izhikevich neurons, replicating spiking dynamics found in the cortex, and 40 million synapses in real time [21], while the flexibility of the platform can be used to explore novel algorithms for learning [22].

A. Interface With Nengo

While with PyNN it is possible to define arbitrary network structures, using the neural engineering framework (NEF) [23], it is possible to encode functions and dynamical systems in networks of spiking neurons. Using the NEF, it is possible to build complex cognitive architectures such as SPAUN [24], a spike-based functional model of the brain that makes comparisons with human neural and behavioral data possible. SpiNNaker has, therefore, been interfaced with Nengo [25], the software that implements the NEF, enabling users to create neural networks by specifying the functions to be computed [13]. Nengo translates the functions into neural circuitry by calculating neuronal and connectivity parameters, while PACMAN distributes and configures the model on the board. Through the use of the NEF, SpiNNaker becomes a "neural computational box": input values and vectors are encoded in spiking activity using the NEF principles directly on the SpiNNaker board. The desired computation is performed in real time by spiking neurons, and output values and vectors are decoded from spiking activity. Interfacing with Nengo shows how different front–ends can be interfaced with PACMAN and how flexibly the platform can be programmed with specialized neural kernels, such as the ones performing the NEF encoding and decoding processes.

B. Interface With AER sensors

Biological inspiration is not confined to the exploration of computational architectures and methods, but is also

extended to neuromorphic [26] sensors. Millisecond-precise pulse encoding has been used to explain the ability of the visual system to process information and to recognize complex, dynamical scenes quickly [27]. With the first observable differences in the temporal lobe starting after 150 ms of the stimulus onset, and with several synaptic stages required to arrive at the infero-temporal cortex (IT, the visual area where object recognition takes place), neurons can emit at most one spike to encode the information, and are believed to encode it in the spike timing [28].

AER sensors can be used to exploit the temporal characteristics of sensory information with event-based approaches. Silicon retinae [29]–[31], for example, take inspiration from their biological counterparts to implement an alternative approach to frame-based image processing on a neuromorphic substrate. Each pixel operates asynchronously, sending an AER message within a few microseconds of a local light intensity change without having to wait for a complete frame to be scanned, resulting in a reduction of latency and redundancy in visual information transmission. For an example showing the benefits of event-based over frame-based systems, see the European Union "Convolution Address Event Representation (AER) Vision Architecture for Real-Time" project [32].

These sensors use native event-based processing and AER representation to encode sensory information, and can, therefore, be interconnected directly to SpiNNaker, which acts as an event-based computing platform. In collaboration with the Instituto de Microelectronica de Seville (Sevilla, Spain) we have connected a silicon retina to SpiNNaker using an FPGA [33], which translates incoming retinal AER events to the self-timed 2-of-7 protocol used by SpiNNaker interchip links, directly injecting spikes (MC packets) into the packet-switched network fabric. Using this mechanism, the sensor is represented on SpiNNaker as a "virtual chip." At the model level, the silicon retina can be instantiated in PyNN as:

```
pol_0, pol_1 = p.instantiate_retina( )
```

creating two populations ("pol_0" and "pol_1", one for each "polarity," encoding increasing and decreasing luminance, respectively) where neurons are topographically organized in a 2-D visual field. These populations produce spikes whenever the silicon retina emits an event, and can arbitrarily be interconnected to other populations in the model. PACMAN automatically maps each population to a specific model instantiation, preserving the connectivity information.

Analogous interfaces with AER sensors have been developed in collaboration with the Institute of Neuroinformatics (Zurich, Switzerland; using the DVS sensor [30] and the "silicon cochlea" [34]), with the Biology Group at the University of Osaka (Osaka, Japan; using a sensor inspired by the sustained and transient responses of the retina [35]), and with the Institute of Vision (Paris, France; using the ATIS silicon retina [36]).

C. Integration With Robotic Platforms

While integration with AER sensors exploits the event-driven nature of the system, interfacing it with robotic platforms in real environments shows SpiNNaker's real-time characteristics.

As with AER sensors, the robotic platform becomes available at the model level using PyNN or Nengo, while the system is configured automatically using PACMAN, enabling message transmission to and from the robot and the sensors through a small customized interface board [37]. The robot is a custom omnidirectional mobile platform, with embedded low-level motor control and elementary sensory systems, developed by the Neuroscientific System Theory group of the Technische Universität München (Munich, Germany). The overall system is a standalone, autonomous, reconfigurable robotic platform with no personal computer in the loop.

We demonstrate a closed perception–action loop in an example where the robot agent has to discriminate between two different stimuli and move toward the preferred one (a "+"), while backing off from the detractor (an "×"). This is a small model that uses less than 10% of the resources on the 48-node board, but it serves to illustrate a number of the capabilities of the system.

The network structure used is represented in Fig. 10. The two populations representing the different polarities of a 128× 128 silicon retina are instantiated, as illustrated in Section IX-B. These populations are connected to four different feature maps, representing the result of the convolution between the retinal input and a kernel represented as the white insert in the four feature maps in Fig. 10, where the black lines represent excitatory connections while the white surround represents inhibitory flanks. This operation, computed in parallel by all feature maps by means of spiking events, is similar to the one performed by the mammalian primary visual cortex, where cells are selectively active accordingly to the stimulus orientation [38], as previously done in a model of visual attention running on a four-node SpiNNaker board [33]. Different feature maps inhibit each other in order to enhance response contrast. The following layer behaves as a local combination of oriented edge detectors, similar to the first layers of the HMAX model, a model of object recognition inspired by the visual cortex [39]. If the "+" is recognized (as a combination of vertical and horizontal edges), the agent is driven forward toward the preferred stimulus; conversely, if an "×" is detected as a combination of $+/-45^\circ$ oriented lines, the robot moves backward.

Robot movements are controlled by the output population, comprising two "motor" neurons (one for moving forward and one for moving backward), represented by the two vertical bars in Fig. 10.

The retina and the robot are accessible through PyNN, which is also used to describe the rest of the network model, performing different steps of visual processing and orienting its response to the location where a preferred stimulus is detected.

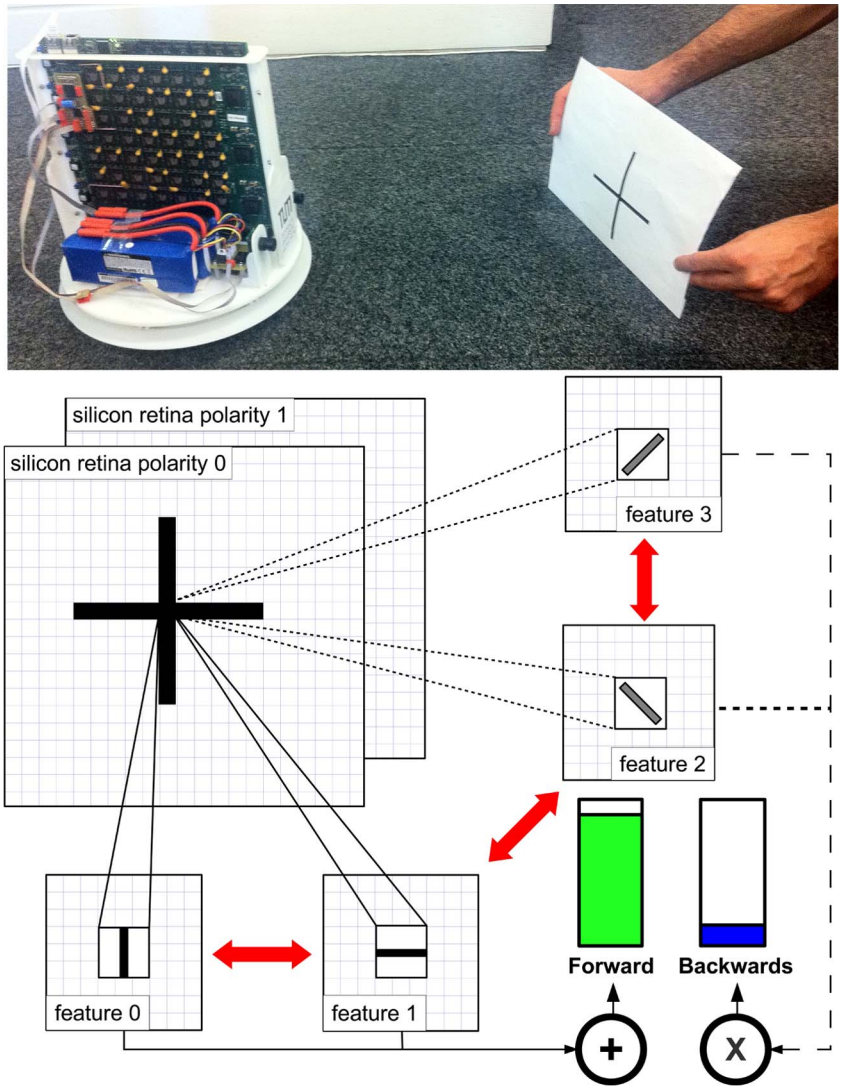

Fig. 10. ***Example robotic closed perception–action loop. A "+" is shown to the robot, which extracts and combines the vertical and horizontal lines, moving forward. Gray kernels and dashed lines represent the fact that the pathways for the "×" detection are not activated, as a "+" is presented.***

X. FUTURE PLANS

Current SpiNNaker hardware has seen use across the computational neuroscience and neurorobotic communities. All of the major hardware functions required to build larger machines have now been developed and tested, and the remaining tasks to build larger machines are now primarily related to the manufacture of further packages and PCBs.

A major commitment over the next two years is to deliver a machine with at least half a million processors as a contribution to the European Union Flagship Human Brain Project (HBP), where SpiNNaker will be one of the neuromorphic "platforms" offered to the wider HBP community.

An earlier, less formal, commitment is to demonstrate the capability of SpiNNaker to support a real-time implementation of the University of Waterloo (Waterloo, ON, Canada) SPAUN model [24]. This is expected to require a system of around 36 48-node SpiNNaker boards, or 30 000 processors, though this estimate should come down with Nengo support for sparse connectivity and reduced firing rates, and will be a solid demonstration of the capability of the SpiNNaker machine as a platform to support large-scale real-time spiking neural models.

XI. RELATED WORK

While SpiNNaker represents a particular combination of digital many-core computing with a lightweight communications infrastructure tuned to modeling large-scale spiking neural networks in biological real time, there are a

number of other designs that take a different approach to achieve similar end goals [40]. These various approaches can be classified according to whether they use digital or analog technology to model the neurons and synapses, the communications topology employed, and the support for synaptic plasticity.

Digital models may be implemented on conventional general-purpose computers, including cluster machines and high-performance computers, or on special-purpose hardware such as FPGAs [41], [42], graphics professor units [43], or custom silicon [44]. Analog models [45] may be subthreshold [46], whereupon biological real-time performance is achievable, or above threshold [47], where the circuits are likely to be much faster than biological real time. Notable large-scale projects include the following.

- The Stanford Neurogrid [46] employs subthreshold analog circuits with digital spanning tree AER communications [48] for real-time neural modeling. Neurogrid can model a million neurons in real time while consuming only 3 W. It combines unicast and multicast digital routing with analog signaling across a local "diffusion network."
- The IBM neurosynaptic core [49] employs custom digital circuits to achieve a one-to-one correspondence between the hardware and software simulation models. It is intended to form a generic cognitive subsystem [44]. It uses AER communication.
- The Heidelberg HICANN system [47] employs wafer-scale above threshold analog circuits that operate at 10^4x biological real time using a two-layer AER protocol, one layer for intrawafer communication and a second layer for interwafer communication.
- The Cambridge BlueHive system [41] employs digital circuits on FPGAs to deliver real-time performance. The communication is not pure AER; multicast is implemented using a set of "fan-out" messages that carry the destination, weight, and delay.

These examples illustrate the diversity of approaches taken to address the problem of modeling large-scale systems of spiking neurons in real time or faster. There are arguments on both sides of the analog/digital divide (for example, energy-efficiency favors analog, whereas flexibility and repeatability favors digital), and on most other design decisions, so the area is still wide open to new ideas, and rather lacking in robust benchmarks that can be used to make quantitative comparisons between alternative approaches.

XII. CONCLUSION

The SpiNNaker project has been 15 years since conception and eight years in (funded) execution. Much time and effort has gone into understanding the brain-modeling problem domain and developing the architecture, silicon, and software infrastructure. While the software development will be ongoing, the architecture and silicon are now working reliably and delivering very much as originally anticipated [1].

The process of delivering the potential of the SpiNNaker platform is now underway, and early indications are largely positive. The platform is proving flexible, relatively easy to use (though there is always room for improvement in this dimension), and capable of delivering useful results across a wide range of application areas.

As the platform is scaled up toward the ultimate million-core machine, new challenges will emerge, particularly in the area of management, application mapping and loading performance, the observability of activity within the machine, and most notably with debugging large-scale models running on the machine. All of these are ongoing areas of research and development, but with help and feedback from a growing (and so far very forgiving) community of users, and secure funding within the HBP alongside a number of other funded projects that will support extensive use of the platform at the University of Manchester [including a European Research Council Advanced Grant and several Engineering and Physical Sciences Research Council (EPSRC)-funded collaborations], we are committed to continued improvement of the capabilities of the platform.

The time is right to scale up our ambition to understand the information processing principles at work in the brain, and the SpiNNaker platform has been designed to deliver a broad capability to support this ambition. The next five years will be crucial in determining the extent to which we can succeed in delivering a platform with the capabilities required to support the global brain research program. ■

Acknowledgment

The SpiNNaker project has benefited from contributions from many people in the team at the University of Manchester, collaborators at the universities of Southampton, Cambridge, and Sheffield, industry partners, and many external collaborators, only some of whom has there been space to mention in the text, but the authors would like to note here the contribution of J. Conradt of the Technische Universität München (Munich, Germany) who developed the robot platform shown in Fig. 10. They particularly wish also to acknowledge the benefits accrued from participation in the Capo Caccia and Telluride neuromorphic workshops, and they are grateful to the organizers for the opportunities for collaborations that have emerged from these workshops. The authors would also like to acknowledge the helpful comments and feedback from the anonymous reviewers of the first draft of this paper.

REFERENCES

[1] S. Furber and S. Temple, "Neural systems engineering," *J. Roy. Soc. Interface*, vol. 4, no. 13, pp. 193–206, 2007.

[2] J. J. Wade, L. J. McDaid, J. Harkin, V. Crunelli, and J. A. S. Kelso, "Bidirectional coupling between astrocytes and neurons mediates learning and dynamic coordination in the brain: A multiple modeling approach," *PLoS ONE*, vol. 6, no. 12, 2011, DOI: 10.1371/journal.pone.0029445.

[3] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*. Boston, MA, USA: Kluwer, 1994.

[4] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst.*, vol. 47, no. 5, pp. 416–434, May 2000.

[5] Top 500 Supercomputers, Jun. 2013. [Online]. http://top500.org/lists/2013/06/

[6] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker system architecture," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.

[7] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocessors Microsyst.*, vol. 35, no. 2, pp. 152–166, 2011.

[8] E. Painkras, L. A. Plana, J. D. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "SpiNNaker: A 1 W 18-core system-on-Chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.

[9] C. Patterson, F. Galluppi, A. D. Rast, and S. B. Furber, "Visualising large-scale neural network models in real-time," in *Proc. Int. Joint Conf. Neural Netw.*, Brisbane, Australia, Jun. 10–15, 2012, DOI: 10.1109/IJCNN.2012.6252490.

[10] F. Galluppi, S. Davies, A. D. Rast, T. Sharp, L. A. Plana, and S. B. Furber, "A hierarchical configuration system for a massively parallel neural hardware platform," in *Proc. 9th Conf. Comput. Front.*, 2012, pp. 183–192.

[11] T. Sharp, L. A. Plana, F. Galluppi, and S. B. Furber, "Event-driven simulation of arbitrary spiking neural networks on SpiNNaker," *Neural Information Processing*, Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 2011, vol. 7064, pp. 424–430.

[12] A. 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. Neuroinf.*, vol. 2, 2008, DOI: 10.3389/neuro.11.011.2008.

[13] T. C. Stewart, B. Tripp, and C. Eliasmith, "Python scripting in the Nengo simulator," *Front. Neuroinf.*, vol. 3, 2009, DOI: 10.3389/neuro.11.007.2009.

[14] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, V. J. Wedeen, and O. Sporns, "Mapping the structural core of human cerebral cortex," *PLoS Biol.*, vol. 6, no. 7, Jul. 2008, DOI: 10.1371/journal.pbio.0060159.

[15] T. Binzegger, R. Douglas, and K. Martin, "A quantitative map of the circuit of cat primary visual cortex," *J. Neurosci.*, vol. 24, no. 39, pp. 8441–8453, Sep. 2004.

[16] O. Gewaltig, M. Diesmann, and M.-O. Gewaltig, "NEST (NEural Simulation Tool)," *Scholarpedia*, vol. 2, no. 4, 2007, DOI: 10.4249/scholarpedia.1430.

[17] D. Goodman, "Brian: A simulator for spiking neural networks in Python," *Front. Neuroinf.*, vol. 2, 2008, DOI: 10.3389/neuro.11.005.2008.

[18] H. Plesser, J. Eppler, A. Morrison, M. Diesmann, and M. O. Gewaltig, "Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers," *Euro-Par 2007 Parallel Processing*, vol. 4641, Berlin, Germany: Springer-Verlag, 2007, pp. 672–681.

[19] E. Stromatias, F. Galluppi, C. Patterson, and S. Furber, "Power analysis of large-scale, real-time neural networks on SpiNNaker," in *Proc. Int. Joint Conf. Neural Netw.*, 2013, pp. 1570–1577.

[20] E. Izhikevich and G. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proc. Nat. Acad. Sci. USA*, vol. 105, no. 9, pp. 3593–3598, Mar. 2008.

[21] T. Sharp, F. Galluppi, A. D. Rast, and S. B. Furber, "Real-time simulation of detailed cortical microcircuits on SpiNNaker," *J. Neurosci. Methods*, vol. 210, no. 1, pp. 110–118, 2012.

[22] S. Davies, F. Galluppi, A. D. Rast, and S. B. Furber, "A forecast-based STDP rule suitable for neuromorphic implementation," *J. Neural Netw.*, vol. 32, pp. 3–14, 2012.

[23] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA, USA: MIT Press, 2003.

[24] C. Eliasmith, T. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[25] F. Galluppi, S. Davies, S. Furber, T. Stewart, and C. Eliasmith, "Real time on-chip implementation of dynamical systems with spiking neurons," in *Proc. Int. Joint Conf. Neural Netw.*, 2012, DOI: 10.1109/IJCNN.2012.6252706.

[26] C. A. Mead, *Analog VLSI and Neural Systems*. Reading, MA, USA: Addison-Wesley, 1989.

[27] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520–522, 1996.

[28] R. Van Rullen and S. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex," *Neural Comput.*, vol. 13, no. 6, pp. 1255–1283, 2001.

[29] C. Mead and M. Mahowald, "A silicon model of early visual processing," *Neural Netw.*, vol. 1, no. 1, pp. 91–97, 1988.

[30] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[31] J. Lenero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6μs latency asynchronous frame-free event-driven dynamic-vision-sensor," *IEEE J. Solid State Circuits*, vol. 46, no. 6, pp. 1443–1455, Jun. 2011.

[32] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, L. Camunas-Mesa, R. Berner, M. Rivas-Perez, T. Delbruck, S. Liu, R. Douglas, P. Hafliger, G. Jimenez-Moreno, A. Civit Ballcels, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "CAVIAR: A 45 k neuron, 5 M synapse, 12 G connects/s AER hardware sensory-processing- learning-actuating system for high-speed visual object recognition and tracking," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[33] F. Galluppi, K. Brohan, S. Davidson, T. Serrano-Gotarredona, J. Carrasco, B. Linares-Barranco, and S. B. Furber, "A real-time, event-driven neuromorphic system for goal-directed attentional selection," in *Neural Information Processing*, vol. 7664, Berlin, Germany: Springer-Verlag, 2012, pp. 226–233.

[34] A. van Schaik and S. Liu, "AER EAR: A matched silicon cochlea pair with address event representation interface," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, pp. 4213–4216.

[35] S. Kameda and T. Yagi, "An analog VLSI chip emulating sustained and transient response channels of the vertebrate retina," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1405–1412, Sep. 2003.

[36] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression," in *Proc. Int. Solid-State Circuits Conf.*, 2010, pp. 400–401.

[37] C. Denk, F. Llobet-Blandino, F. Galluppi, L. Plana, S. Furber, and J. Conradt, "Real-time interface board for closed-loop robotic tasks on the SpiNNaker neural computing system," *Artificial Neural Networks and Machine Learning—ICANN 2013*, vol. 8131, Berlin, Germany: Springer-Verlag, 2013, pp. 467–474.

[38] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture of the cat's cortex," *J. Physiol.*, vol. 160, pp. 106–154, 1962.

[39] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 411–426, Mar. 2007.

[40] R. Cattell and A. Parker, "Challenges for brain emulation: Why is it so difficult?" *Natural Intell.*, vol. 1, no. 3, pp. 17–31, 2012.

[41] P. J. Fox, S. W. Moore, 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. IEEE 20th Int. Symp. Field-Programmable Custom Comput. Mach.*, 2012, pp. 133–140.

[42] A. S. Cassidy, J. Georgiou, and A. G. Andreou, "Design of silicon brains in the nano-CMOS era: Spiking neurons, learning synapses and neural architecture optimization," *Neural Netw.*, vol. 45, pp. 4–26, Jun. 2013.

[43] A. K. Fidjeland and M. P. Shanahan, "Accelerated simulation of spiking neural networks using GPUs," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2010, DOI: 10.1109/IJCNN.2010.5596678.

[44] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez-Icaza, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. Rubin, R. Manohar, and D. S. Modha, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 2012, DOI: 10.1109/IJCNN.2012.6252637.

[45] G. Indiveri, B. Linares-Barranco, T. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S. C. Liu, P. Dudek, P. Hafliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Front. Neurosci.*,

vol. 5, no. 23, May 2011, DOI: 10.3389/fnins.2011.00073.

[46] R. Silver, K. Boahen, S. Grillner, N. Kopell, and K. Olsen, "Neurotech for neuroscience: Unifying concepts, organizing principles, and emerging tools," *J. Neurosci.*, pp. 11807–11819, Oct. 2007.

[47] J. Schemmel, D. Bruderle, A. Grubl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. Int. Symp. Circuits Syst.*, 2010, pp. 1947–1950.

[48] P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, and K. Boahen, "A multicast tree router for multichip neuromorphic systems," *IEEE Trans. Circuits Syst.*, 2013, DOI: 10.1109/TCSI.2013.2284184.

[49] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45 pj per spike in 45 nm," in *Proc. Custom Integr. Circuits Conf.*, Sep. 2011, DOI: 10.1109/CICC.2011.6055294.

ABOUT THE AUTHORS

Steve B. Furber (Fellow, IEEE) was born in Manchester, U.K., in 1953. He received the B.A. degree in mathematics and the Ph.D. degree in aerodynamics from the University of Cambridge, Cambridge, U.K., in 1974 and 1980, respectively, and honorary doctorates from Edinburgh University, Edinburgh, U.K., in 2010 and Anglia Ruskin University, Cambridge, U.K., in 2012.

From 1978 to 1981, he was Rolls Royce Research Fellow in Aerodynamics at Emmanuel College, Cambridge, U.K., and from 1981 to 1990, he was at Acorn Computers Ltd., Cambridge, U.K., where he was a principal architect of the BBC Microcomputer, which introduced computing into most U.K. schools, and the ARM 32-bit RISC microprocessor, over 40 billion of which have been shipped by ARM Ltd.'s partners. In 1990, he moved to the ICL Chair in Computer Engineering at the University of Manchester, Manchester, U.K., where his research interests include asynchronous digital design, low-power systems on chip, and neural systems engineering.

Prof. Furber is a Fellow of the Royal Society, the Royal Academy of Engineering, the British Computer Society, the Institution of Engineering and Technology and the Computer History Museum (Mountain View, CA). He was a Millennium Technology Prize Laureate (2010) and holds an IEEE Computer Society Computer Pioneer Award (2013).

Francesco Galluppi received the B.Sc. degree in electronic engineering from the University of Rome "Roma Tre," Rome, Italy, in 2005, the M.Sc. degree in cognitive psychology from the University of Rome "La Sapienza," Rome, Italy, in 2010, and the Ph.D. degree from the University of Manchester, Manchester, U.K., in 2013, working on neurally inspired hardware systems and sensors.

He visited the Departamento de Tecnología Electrónica, University of Málaga, Málaga, Spain, in 2007, working on robotic assistive technologies. His interests lie in studying the biological basis of human behavior, and how technology interacts with it.

Steve Temple received the B.A. degree in computer science and the Ph.D. degree for research into local area networks from the University of Cambridge, Cambridge, U.K., in 1980 and 1984, respectively.

He was subsequently employed as a Research Fellow at the University of Cambridge Computer Laboratory. He was a self-employed computer consultant from 1986 to 1993 when he took up his current post of Research Fellow in the School of Computer Science, University of Manchester, Manchester, U.K.

Luis A. Plana (Senior Member, IEEE) received the Ingeniero Electrónico degree (*cum laude*) from Universidad Simón Bolívar, Caracas, Venezuela, in 1978, the M.Sc. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1980, and the Ph.D. degree in computer science from Columbia University, New York, NY, USA, in 1998.

He was with Universidad Politécnica, Venezuela, for over 20 years, where he was a Professor of Electronic Engineering. Currently, he is a Research Fellow with the School of Computer Science, University of Manchester, Manchester, U.K. His research interests include embedded system design and on-chip interconnect.