

# An efficient SpiNNaker implementation of the Neural Engineering Framework

Andrew Mundy and James Knight

School of Computer Science,  
University of Manchester,  
M13 9PL, UK

Email: andrew.mundy@ieee.org

Email: james.knight@manchester.ac.uk

Terrence C. Stewart

Centre for Theoretical Neuroscience,  
University of Waterloo,  
Waterloo, ON,

Canada N2L 3G1

Email: tcestewar@uwaterloo.ca

Steve Furber

School of Computer Science,  
University of Manchester,  
Oxford Road, Manchester,  
M13 9PL, UK

Email: steve.furber@manchester.ac.uk

**Abstract**—By building and simulating neural systems we hope to understand how the brain may work and use this knowledge to build neural and cognitive systems to tackle engineering problems. The Neural Engineering Framework (NEF) is a hypothesis about how such systems may be constructed and has recently been used to build the world's first functional brain model, Spaun. However, while the NEF simplifies the design of neural networks, simulating them using standard computer hardware is still computationally expensive – often running far slower than biological real-time and scaling very poorly: problems the SpiNNaker neuromorphic simulator was designed to solve. In this paper we (1) argue that employing the same model of computation used for simulating general purpose spiking neural networks on SpiNNaker for NEF models results in sub-optimal use of the architecture, and (2) provide and evaluate an alternative simulation scheme which overcomes the memory and compute challenges posed by the NEF. This proposed method uses factored weight matrices to reduce memory usage by around 90% and, in some cases, simulate 2000 neurons on a processing core – double the SpiNNaker architectural target.

## I. INTRODUCTION

For a given power budget, two factors limit the simulation of neural networks on any computing platform: scale and time. Theoretically, any scale of network may be simulated but as scale increases simulation time follows. Conversely, if the simulation time is limited (for example, if biological real-time is necessary) then only a limited scale of network may be simulated. Specialised *neuromorphic* hardware tries to avoid these constraints by parallelising and distributing computational effort and relying on dense interconnection of the computing elements. The SpiNNaker platform [1] is one of a range of neuromorphic simulators (including Neurogrid [2], BrainScaleS [3] and TrueNorth [4]) which should benefit researchers of large-scale neural models.

The Neural Engineering Framework (NEF) [5] is a hypothesis about how neurons may be used to encode abstract mathematical constructs, such as scalars and vectors, that we often use to model the real world. Its successes so far include the Spaun model of cognition [6] and a spiking neural

network that encodes and decodes the main lexical relations in WordNet [7]. As with all neural systems, the NEF has proven costly to simulate: the Spaun model typically required 2.5 h of compute time for 1 s of simulation [8, §V]. Two aspects of NEF networks that make them particularly costly to simulate are the high firing rates of individual neurons (often up to 400 Hz) and the dense synaptic matrices used to connect neuronal populations. The former presents a significant communication cost to any specialised neuromorphic hardware and the latter requires that large amounts of memory be used to represent the neural network with all the associated costs of transferring large blocks of data that implies.

In this paper we:

- 1) Argue that due to the properties of the NEF, the existing solutions and algorithms used to simulate neural networks on SpiNNaker will not scale satisfactorily to large-scale models such as Spaun.
- 2) Detail a method by which features of the NEF may be used to reduce the memory and compute costs associated with its simulation.

We hope to use the result to run the full Spaun model in biological real-time.

## II. BACKGROUND

In this section we briefly discuss the SpiNNaker platform and how neural networks are currently simulated on it before introducing the Neural Engineering Framework (NEF) and discussing how models built with it may act to stress SpiNNaker in various ways.

### A. The SpiNNaker platform

The SpiNNaker platform is a massively parallel architecture designed to simulate neural networks. A SpiNNaker machine is constructed from a number of SpiNNaker chips, each connected to their six immediate neighbours using a chip-level interconnection network with a toroidal, triangular mesh topology. Each SpiNNaker chip contains 18 ARM processing cores connected, via a network-on-chip, to each other and the external network through a multicast router. Each core has two small tightly-coupled memories: 32 KiB for instructions and

This work is partially supported by EPSRC grant EP/G015740/1 (BIMPA); the European Union under grant nos. ERC-320689 (BIMPC), FP7-287701 (BrainScales-Extension), and FP7-604102 (HBP); and by the U.S. Office of Naval Research.

64 KiB for data; and shares 128 MiB of off-chip SDRAM with the other cores on the SpiNNaker chip.

SpiNNaker is an event-driven, message-passing computing architecture. The software running on a core may transmit packets to other processing cores to indicate the occurrence of events or to share data. A packet consists of a 32 bit key, used to direct the packet around the network and, optionally, a 32 bit data payload. When a packet reaches a router, its key is inspected to determine which (if any) of the 18 processors and six external links attached to the router it should be forwarded to. On receipt of a packet, a core executes a *callback* function which may inspect the packet and schedule further execution as required.

### B. Simulating neural nets on SpiNNaker

When simulating neural nets on a SpiNNaker machine each core is responsible for simulating a number of point neurons (in the order of a few hundred). When one of these neuron spikes, it transmits a packet whose key uniquely identifies the neuron (for this it requires no payload). This “spike” packet is then routed across the network fabric to the processing cores responsible for simulating each of the neurons that are connected to the firing neuron. On receipt of a “spike” packet a core retrieves the row of the connectivity matrix associated with the firing neuron from SDRAM. Each of these rows describes the synaptic weights and delays associated with the connections between the firing neuron and those simulated on the core. Once a row is retrieved the weights are inserted into an input ring-buffer where they remain until the synaptic delay has elapsed and they are applied to the neuronal input current.

Using this system there are two primary constraints on the number of neurons that may be simulated on a single processing core:

- 1) The amount of memory required to store the synaptic weight matrices must fit within the space available to the core. This is 8 MiB as each core is allocated  $\frac{1}{16}$  of the 128 MiB SDRAM.
- 2) As the majority of processing time is spent in the synaptic processing pipeline, there must be sufficient time for the core to process all incoming ‘spike’ packets; and retrieve and process the synaptic rows during one simulation time-step (see [9]). This time is a function of both the number of spikes received per time-step and the density of the synaptic matrix. Sharp and Furber [9, §III.C] indicate that at most there may be 5000 synaptic events per millisecond when running at biological real-time, where a single synaptic event indicates one spike being passed through a single synapse to a neuron on the receiving core.

These constraints may be satisfied by either allocating fewer neurons to each processing core or by increasing the processing time used for each simulation time-step. Hard time constraints are necessary when SpiNNaker is required to run in biological real-time, as it is in experiments with other neuromorphic hardware. Processor constraints are present for

those with access to only small SpiNNaker machines such as those mounted in mobile robots.

It should be noted that time is also a factor *prior* to the start of any simulation. All data required by the SpiNNaker machine during simulation must be transmitted to it through an ethernet interface, meaning that if more data is required on the machine more time is required to prepare it for simulation. Sharp and Furber [9] note that this preparation time can be of the order of several minutes – something that is undesirable if a real-time simulator is desired.

### C. The Neural Engineering Framework

The Neural Engineering Framework (NEF) extends the concept of “preferred-direction vectors” [10] to all neural populations. Each population represents a vector within a particular space, within which, the firing rate of each neuron reflects the similarity of the represented vector to the neurons “encoding” vector. Using the notation of Stewart and Eliasmith [8] this “encoding” of a variable in vector form into a neuronal response may be expressed as:

$$\delta_i(\mathbf{x}) = G_i [\alpha_i \mathbf{e}_i \cdot \mathbf{x} + J_i^{bias}] \quad (1)$$

Which states that the firing response of neuron  $i$  ( $\delta_i$ ) to the represented value ( $\mathbf{x}$ ) is the response of the neuron model ( $G_i$ ) to an input consisting of a randomly selected gain term ( $\alpha_i$ ), the encoding vector for the neuron ( $\mathbf{e}_i$ ) and a fixed bias current ( $J_i^{bias}$ ). Correspondingly, “decoding” allows a transformation from the spiking actions of neurons into the domain of vectors. Again using the notation of Stewart and Eliasmith [8] we can express this decoding process as:

$$\hat{\mathbf{x}} = \sum_{i=1}^N a_i(\mathbf{x}) \mathbf{d}_i \quad (2)$$

Where the estimate of the original represented value ( $\hat{\mathbf{x}}$ ) is the sum of the spiking activity of each neuron ( $a_i$ ) multiplied by the linear decoder for the neuron ( $\mathbf{d}_i$ ). The decoding vectors may be selected to compute a function of the value represented by the population. Fig. 1 illustrates the encoding of a two-dimensional value using four neurons – the role of the encoding vectors can be seen in that each neuron becomes active for only a small range of the input space.

For a connection between a pair of populations, a (dense) synaptic weight matrix can be calculated by computing the matrix product of the decoders of the pre-synaptic population and the encoders of the post-synaptic population [8]:

$$\omega_{ij} = \alpha_j \mathbf{d}_i \mathbf{e}_j \quad (3)$$

With  $i$  indexing neurons in the pre-synaptic population and  $j$  those in the post-synaptic population.

An illustrative model that we will use later in this paper is the *communication channel*. A communication channel consists of two populations connected with the synaptic weights chosen such that the second ensemble will represent the same value as the first ensemble. The concept is illustrated in

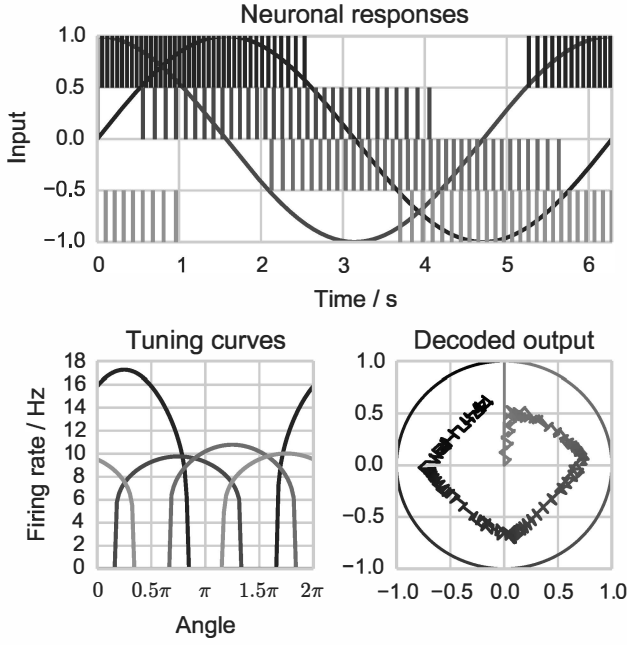


Fig. 1. Representing a 2-dimensional value using four neurons. The input values and spiking responses of the neurons are shown in the top plot. The bottom left-hand plot shows how the firing responses vary with the angle of the 2-D input vector (tuning curves) and the bottom right-hand plot shows a decoding of the population’s representation along with the input value.

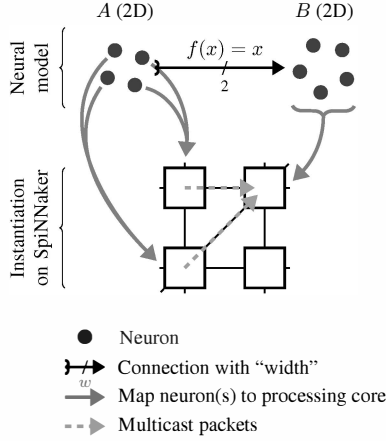


Fig. 2. A NEF communication channel consisting of two populations of neurons and a connection. The diagram indicates how the abstract communication channel might be instantiated on a simplified representation of the SpiNNaker architecture: Population A is split between processing two cores, consequently two streams of multicast packets allow simulation of the communication channel.

Fig. 2; where, to illustrate how computational constraints may be satisfied, the pre-synaptic population has been partitioned across two processing cores.

#### D. Assessing the Neural Engineering Framework (NEF)

It is entirely possible to translate models built using NEF principles into populations of point neurons connected with dense synaptic weight matrices. These networks could then,

in theory, be simulated using the standard SpiNNaker spiking neural network simulator described in §II-B. In this section we show that, based on parameters used in the Spaun functional brain model, this approach would result in sub-optimal usage of a SpiNNaker machine, given the constraints outlined in §II-B.

The Spaun model is built using the Semantic Pointer Architecture (SPA) [11] which uses randomly chosen unit vectors to represent basic concepts. These concepts can then be combined into symbol structures using addition (+) and circular convolution ( $\otimes$ ) operators, both of which can be evaluated using the principles of the NEF. The circular-convolution operator is non-linear and, Eliasmith [11] demonstrates, requires around 70 neurons per dimension to calculate accurately.

For example, using the basic concepts (**MOUSE**, **CHEESE**, **SUBJECT**, **OBJECT**, **EAT**, **VERB**), a semantic pointer representing the sentence “Mice eat cheese” can be formed as  $\mathbf{S} = \mathbf{MOUSE} \otimes \mathbf{SUBJECT} + \mathbf{EAT} \otimes \mathbf{VERB} + \mathbf{CHEESE} \otimes \mathbf{OBJECT}$ . The subject of this sentence could then be extracted using  $\mathbf{S} \otimes \mathbf{SUBJECT}' \approx \mathbf{MOUSE}$  (where the  $'$  operator is an approximate inverse [11, §D.2], calculated using a linear involution operation). However, the result of this operation will not be exactly **MOUSE** – to extract the **MOUSE** semantic pointer, the result needs to be “cleaned up”. This process is performed using an auto-associative memory network [12] the properties of which dictate that, in order to reliably represent a human-scale lexicon, semantic pointers must be drawn from a vector space with around 500 dimensions.

On the basis of these two parameters (500 dimensions and 70 neurons per dimension) we can determine that semantic pointers with this dimensionality would need to be represented using populations of  $3.5 \times 10^4$  neurons. A connection between two of these populations would therefore require a dense synaptic matrix with  $1.225 \times 10^9$  entries. As synaptic weights on SpiNNaker are typically represented as 16 bit values, this full matrix would occupy approximately 2.28 GiB of memory meaning that, as each core only has 8 MiB of SDRAM, the post-synaptic ensemble would have to be distributed amongst 292 cores. This would reduce the number of neurons simulated per core to only 120 – an order of magnitude short of SpiNNaker’s architectural target of 1000 [13].

Representational errors can be reduced both by increasing the number of neurons and by allowing them to fire at a higher rate [5]. However, models built using the NEF typically opt to use fewer neurons firing at rates of up to 400 Hz, meaning that in networks such as the communication channel example discussed in §II-C, each neuron fires at an average rate of around 100 Hz. If this example were simulated with 70 neurons per dimension, the overall spike rate would increase linearly with dimensionality as shown in Fig. 3. In the 4 dimensional case, this means that the post-synaptic ensemble will receive around 20 incoming spikes per time-step which, due to the dense connectivity matrices, will trigger a synaptic event for each of the 280 neurons in the ensemble. This results in a total of around 5600 synaptic events per time-step – exceeding the limit of 5000 events per time-step found by

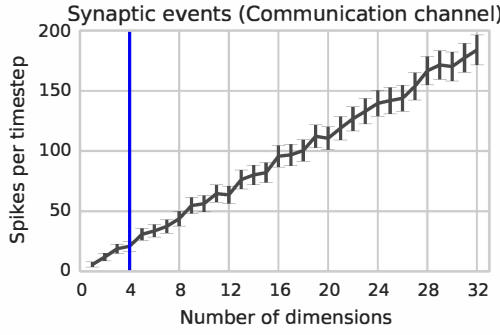


Fig. 3. Spike rates for the pre-synaptic population of a communication channel. The vertical line indicates the point at which 5000 synaptic events per core has been passed. Each population had 70 neurons per dimension.

Sharp and Furber [9].

### III. EXPLOITING FEATURES OF THE NEF FOR EFFECTIVE SIMULATION ON SPINNAKER

In this section we review an alternative simulation scheme that will meet the constraints of the SpiNNaker hardware for a large range of networks constructed using the NEF. The details of the NEF described above suggest this alternate, more efficient, implementation for SpiNNaker. We note that the synaptic weight matrices are computed via (3). This means that all the synaptic weight matrices are exactly *factorable*. Moreover every set of connection weights coming into an ensemble will have the same  $\alpha_j \mathbf{e}_j$  factor applied to it. We can thus *split* the synaptic connection weight matrices into two parts: the “encoder”  $\alpha_j \mathbf{e}_j$  and the “decoder”  $\mathbf{d}_i$ . These matrices are of size  $N \times D$  and  $D \times N$ , where  $D$  is the “dimensionality” of the representation and is generally much smaller than  $N$ , the number of neurons. We can store the encoder in the core that is simulating the post-synaptic population, and we can store the decoder with the pre-synaptic population. This saves significant memory since the two factors are much smaller than the original weight matrix and we only need to store one encoder no matter how many sets of input connections there are. However, since the connection weight matrix is split in this way, we can no longer send a packet to represent each spike. Rather, the spikes are multiplied by the decoder and the resulting vector value is transmitted. Importantly, since the original weight matrices are perfectly factorable, the result is identical to the original SpiNNaker approach in terms of the neural behaviour, but with significantly reduced memory requirements, and, as we shall show, improved processor utilisation. Fig. 4a illustrates the scheme with the full weight matrix stored in the memory of the post-synaptic core and Fig. 4b illustrates the proposed scheme with the factored matrix split across the pre- and post-synaptic cores and with “value”-packets being transmitted between the processors.

We split the execution time of a processing core into three steps: *input filtering*, *neuron update* and *output*. In the “spike” simulation scheme the “inputs” to a processing core take the form of “spike” packets which are used to drive a pipeline

that retrieves synaptic weights from SDRAM; in our proposed scheme the inputs take the form of a set of packets whose payloads can be combined to form a vector. For example, the processing core simulating population  $B$  from Fig. 2 will receive two packets per time-step from each of the two cores simulating population  $A$  which, combined, will form a 2-D vector representing the current decoding of the activity of  $A$ . At the start of each simulation step the reconstructed vector is filtered using the appropriate synaptic filter model to provide the input current to the population – this is the *input filtering* step.

During the *neuron update* step the state of each neuron is updated in turn. First the input for the neuron is computed by calculating the dot product between the encoding vector for the neuron and the current input to the population ( $\alpha_i \mathbf{e}_i \cdot \mathbf{x} + J_i^{bias}$  from (1)). Then the neuron itself is simulated, for example using the Euler method to compute the next state of a LIF neuron. If, during this process, a neuron spikes then its decoding vector ( $\mathbf{d}_i$ ) is looked up and added to a buffer containing the current decoding of the population activity (as in (2)). Once all the neurons have been simulated the output buffer will contain a vector which represents the weighted decoding of any spikes which occurred during the simulation step.

During the *output* stage each element of the vector contained in the output buffer is transmitted in the payload of a multicast packet whose key uniquely identifies the population and element index. The communication fabric routes these “value” packets to cores which simulate connected populations of neurons.

Some ensembles may have multiple outgoing connections with different transformations or functions, meaning that there are multiple decoder matrices to apply when decoding the activity of the population. In these cases we combine the matrices such that the decoding vector for any neuron is the concatenation of the required decoders (i.e.,  $\mathbf{d}_i = [\mathbf{d}_i^{f(x)}, \mathbf{d}_i^{g(x)}, \dots]$ ).

## IV. RESULTS

### A. Processor utilisation

To measure the CPU load when running the algorithm described in §III on SpiNNaker, we developed a simple tool to profile SpiNNaker executables. In this section we use this tool to analyse how the load on the processors simulating the ensembles in the communication channel network shown in Fig. 2 varies with number of dimensions and neurons. Fig. 5 shows how the proportion of CPU time spent in the different phases of the algorithm outlined in §III varies with both dimensionality and neuron count.

The *input filtering* and *output* phases of the algorithm both transmit and receive one packet per simulation time-step for every dimension. Thus the CPU time spent in both phases is predominantly a function of the number dimensions,  $D$ . We built simple models of the CPU cycles spent in these phases

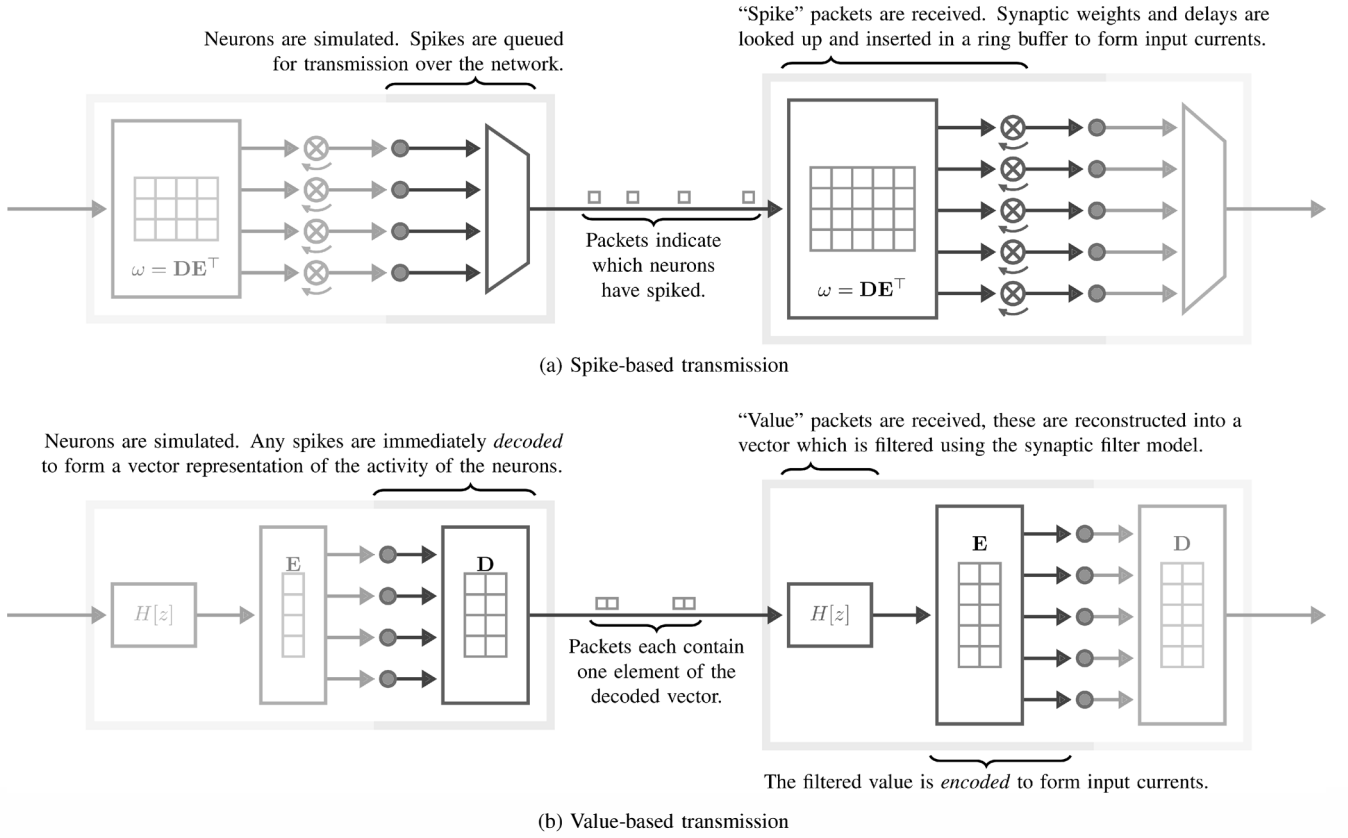


Fig. 4. A comparison of the (a) spike-based and the (b) proposed value-based algorithms.

by fitting the following linear functions to the profiling data using a minimisation of the mean-squared error:

$$c_{\text{input}} = 245 + 43D \quad (4)$$

$$c_{\text{output}} = 100 + 702D \quad (5)$$

Where  $c_{\text{input}}$  and  $c_{\text{output}}$  are measured in cycles, 200 000 of which are available per 1 ms simulation time-step when the processor is running at 200 MHz. During the *neuron update* phase, the dot product of the  $D$ -dimensional input vector and the  $D$ -dimensional encoder vector is computed for each of the  $N$  neurons. Additionally, when a neuron spikes, it is decoded by adding its  $D$ -dimensional decoding vector to the current output. Consequently the compute requirement of the *neuron update* stage is a function of the both the dimensionality and the number of neurons. To model this we again fitted a simple, 1st order, 2-D function to the profiling data by minimising the mean-squared error:

$$c_{\text{neuron}} = 188 + 69N + 13ND \quad (6)$$

Adding these equations together we can model the total CPU cycle count as:

$$c_{\text{total}} = 533 + 745D + 69N + 13ND \quad (7)$$

By substituting  $D = 1$  into (7), we can see that the maximum number of neurons that could be supported in the 1-D case is 2423. This matches our experimental findings that our

new simulator can simulate over 2000 neurons – double the SpiNNaker system’s architectural target of 1000 [13]. In order to compare this model with the spike-based communication model discussed in §II-D, we can further simplify (7) by substituting  $D = \frac{N}{70}$  to reflect Eliasmith’s [11] analysis:

$$c_{\text{total}} = 533 + 80N + 0.19N^2 \quad (8)$$

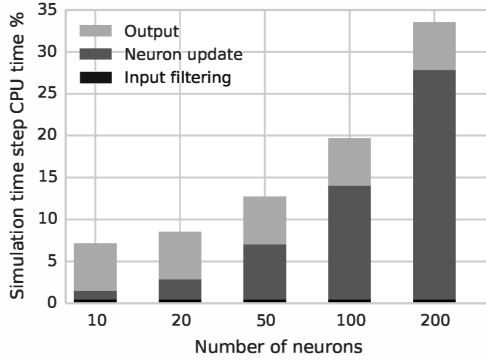
Finally, using the CPU cycle estimates for updating neuronal state and processing incoming synaptic events quoted by Sharp and Furber [9] and the incoming spike rates measured in §II-D, we can build a similar model of the CPU cycle count of the standard SpiNNaker simulator:

$$c_{\text{spike-based-total}} = 128N + 3N^2 \quad (9)$$

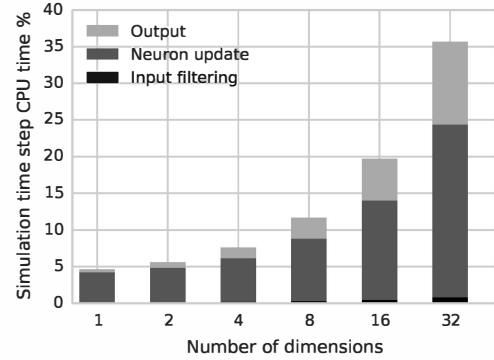
These models show that our implementation requires significantly fewer CPU cycles per neuron than the general purpose SpiNNaker simulator, allowing it to simulate up to 834 neurons per core in this configuration. This is more than  $3.5\times$  the 236 neurons per core that the general purpose SpiNNaker simulator can achieve in the same configuration.

### B. Memory utilisation

The dense connection weight matrix for any connection between populations of  $N_{\text{pre}}$  and  $N_{\text{post}}$  neurons in the NEF will be of size  $N_{\text{pre}} \times N_{\text{post}}$  and, under the spike based algorithm, would be stored entirely in the memory of the post-synaptic core. When factored weight matrices are substituted



(a) 16-D



(b) 100 neurons per population

Fig. 5. Mean percentage of 1ms simulation time step spent in different phases of our simulation algorithm when simulating communication channels of varying dimensionality using different numbers of neurons.

TABLE I  
MEMORY USAGE OF SYNAPTIC WEIGHTS IN BASAL GANGLIA MODEL

Inputs	16	32	64	128
Full weight matrix / MiB	5.38	20.34	78.96	311.04
Factored / MiB	0.18	0.63	2.37	9.18
Reduction / %	96.71	96.89	96.99	97.05

Memory usage of the synaptic weights (full matrix and factored) for the standard Basal Ganglia implementation from Nengo (70 neurons per input).

for the full weight matrix, as described in §III, the memory usage is split between the pre- and post-synaptic cores. The post-synaptic core needs to store a single encoder matrix of size  $D_{\text{post}} \times N_{\text{post}}$  while the pre-synaptic core will store a decoder matrix of size  $D_{\text{pre}} \times N_{\text{pre}}$ , where  $D_{\text{population}}$  is the dimensionality of the population.

For example, a 16-D communication channel constructed from two 1120-neuron populations will require 1 254 400 values to be stored in the memory of the post-synaptic core to represent the full weight matrix. If factored weight matrices are used then both the pre- and post-synaptic cores must store 17 920 values, although the post-synaptic core will not need to store any more data for further incoming connections. Using 16 bit to store each synaptic value and, as in our implementation, 32 bit to store each encoder/decoder value this becomes 2.4 MiB to store the full weight matrix or 70.0 KiB each to store the encoder and decoder (a total of 140.0 KiB). Factored weight matrices achieve a saving of 94.3%.

Table I shows the memory usage from the connections within the basal ganglia model used in Spaun. The reduction in memory usage that results from using factored weight matrices is significant – indeed, it should be noted that for a basal ganglia accepting 128 inputs representing the connection weights would require the SDRAM of three SpiNNaker chips if full weight matrices were used.

### C. Simulating large-scale models

To illustrate the correctness of our technique and to provide a comparison between SpiNNaker and the reference NEF

implementation (Nengo [14]), we simulated a larger model based on the basal ganglia model used in Spaun [15].

For the purposes of our benchmark, we assigned a 16-D semantic pointer to each basal ganglia input and generated utility values over time by comparing these to a repeating sequence of semantic pointers. A sample of the basal ganglia input and output are shown in Fig. 6.

We simulated this model for 10s using a SpiNNaker machine and the standard NEF implementation on a standard desktop PC (3 GHz AMD Athlon II X3 445). Fig. 7 shows the time required to perform certain stages of preparing, uploading data to the SpiNNaker machine, executing the model and retrieving data for two scales of model. As expected, SpiNNaker is able to perform the simulation in biological real-time and is thus significantly faster than Nengo for this example despite the additional overheads. It is likely that as the scale and complexity of models increase the overheads will follow, but this should be less than the expected growth in simulation time on the PC.

## V. DISCUSSION

### A. Comparison to prior SpiNNaker implementations

Galluppi, Davies, Furber, *et al.* [16] demonstrated a SpiNNaker implementation of the NEF using the approach described in §II-B. For the reasons advanced in this paper, we believe that this system would not have been capable of successfully simulating models of the scale or complexity of Spaun in biological real-time. First hand experience with this system indicated that the load time required to transfer full synaptic matrices to a SpiNNaker machine could be extensive. To avoid this Galluppi *et al.* suggested transmitting factored weight matrices and performing their multiplication on SpiNNaker. While this would have reduced the load time it may have resulted in poor accuracy due to the lack of floating point hardware on SpiNNaker. Moreover, it would have neither reduced the amount of memory that would be required on SpiNNaker nor reduced the overloading of the processors due to high spike rates and would still have resulted in inefficient use of the architecture.

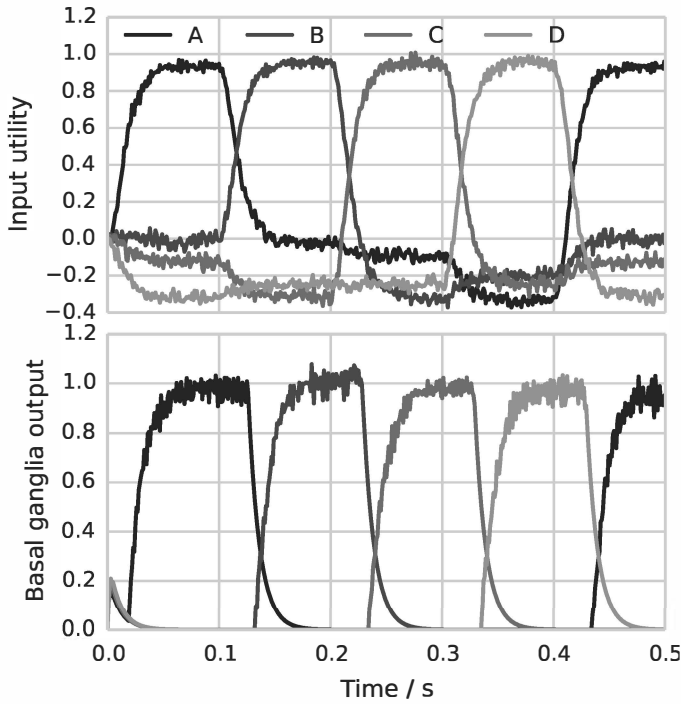


Fig. 6. SpiNNaker simulation of the Basal Ganglia model, selecting between 4 inputs labelled A, B, C and D, presented to it in sequence.

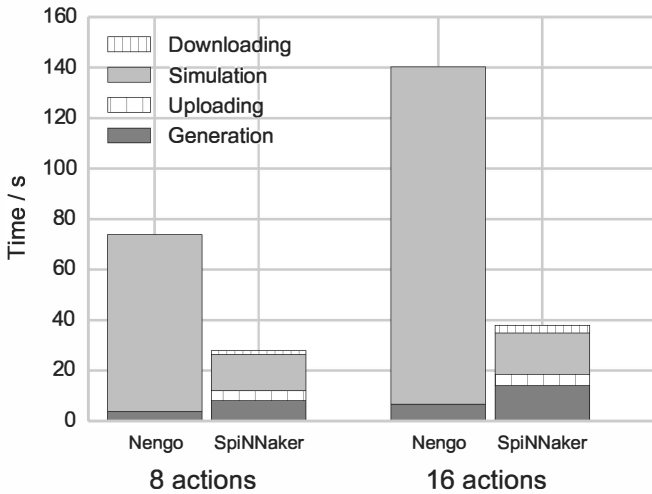


Fig. 7. The time spent in different stages of both the SpiNNaker and the Nengo reference simulators when running the basal ganglia benchmark.

### B. Comparison to other neuromorphic simulators

SpiNNaker is one of a range of neuromorphic simulators, a family including both analogue (e.g., Neurogrid [2], BrainScaleS [3]) and digital (e.g., TrueNorth [4]) hardware, which aim to reduce the power consumption and execution time for simulating large neural models. These simulators can be expected to require less power than SpiNNaker for a given scale of model [17], but are limited to simulating the types

of neurons and synapses they were fabricated with, whereas SpiNNaker can be reprogrammed to investigate new models. In the context of running NEF models of the type discussed in this paper, there is another key limitation of these systems – the number of hardware synapses associated with each neuron. In the case of TrueNorth this number is only 256 and for the BrainScaleS 16 000, restricting the size of the populations that can be connected using dense connectivity matrices to only 16 and 126 respectively.

### C. Comparison to general purpose computer simulators

The reference implementation of the NEF is Nengo [14]. This defines a backend-agnostic set of elements that can be used to construct networks using the principles of the NEF, and a simulation tool which uses the same technique of factoring weight matrices that we have exploited here to reduce memory usage. Bekolay, Bergstra, Hunsberger, *et al.* [14] show that this reference implementation performs well in comparison to alternative PC-based neural simulators such as Brian [18], NEURON [19] and NEST [20] and can also be accelerated using commodity GPU hardware. However, while an ATI Radeon HD 7970 GPU can simulate around 500 000 000 neurons in real-time [14] – comparable to our approach running on a 48 chip SpiNNaker system – scaling beyond one GPU is likely to be constrained by PCI bus bandwidth. Furthermore, GPUs can be expected to consume significantly more power [21] than an equivalent SpiNNaker system [17].

We have extended the work of Bekolay *et al.* by indicating specific reductions in memory usage and by transferring the factored weight matrix simulation scheme to a distributed computing architecture. A particular advantage of the SpiNNaker architecture is that it will simulate networks in biological real-time regardless of model scale and complexity (see Fig. 7) – something that is not possible for commodity hardware. Furthermore, as a real-time system it is able to interface with neuromorphic hardware (e.g., artificial retinas [22], or cochleas [23]).

### D. General applicability of the algorithm

As SpiNNaker is an example of a message passing architecture we expect that our simulation scheme, transmitting decoded representations of ensemble activity rather than spike activity, to be equally applicable to simulating neural nets on any message passing distributed architecture, such as MPI.

Additionally, beyond the NEF, factored weight matrices may be applicable to further types of synaptic matrix. Furthermore, while we have specifically optimised for the case where the factorisations of all connections terminating at a given population may share the same encoder matrix, this need not be the case.

### E. Future work

This work has laid the basis for simulating truly large-scale NEF models in biological real-time. We plan several improvements which should allow a greater number of neurons

to fit on a single core and, beyond this, we anticipate the implementation of further neuron and synapse models. We also intend to better assess the computational cost or gain of value-based transmission when implementing learning rules (e.g., [24]) and investigate how a constant traffic pattern affects the SpiNNaker network architecture.

## VI. CONCLUSION

The dense synaptic weight matrices and high firing rates characteristic of neural networks built using the NEF lead to inefficient use of the SpiNNaker architecture when using the standard algorithms for simulating neural nets. In particular storing the synaptic matrices of these networks requires large amounts of memory and the high firing rates and dense neural connectivity exceed the computational resources available to a SpiNNaker core running in biological real-time. To overcome these constraints we extended a simulation scheme proposed by Bekolay, Bergstra, Hunsberger, *et al.* [14] which used factored weight matrices. This proposed scheme requires around 90% less memory in many cases and is able to simulate many more neurons per core than would otherwise be possible – up to 2000 neurons per core, double the SpiNNaker architectural target. We intend to use the algorithm we have presented to simulate the Spaun functional brain model in biological real-time.

## ACKNOWLEDGEMENTS

The authors would like to extend their thanks to the organisers of the Telluride Neuromorphic Cognition Engineering Workshop.

## REFERENCES

- [1] S. Furber, F. Galluppi, S. Temple, and L. Plana, “The SpiNNaker project,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014, ISSN: 0018-9219.
- [2] B. Benjamin, P. Gao, E. McQuinn, *et al.*, “Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014, ISSN: 0018-9219.
- [3] 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 *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, IEEE, 2010, pp. 1947–1950.
- [4] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
- [5] C. Eliasmith and C. H. Anderson, *Neural Engineering*. MIT Press, 2004, ISBN: 978-0262550604.
- [6] C. Eliasmith, T. C. 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.
- [7] E. Crawford, M. Gingerich, and C. Eliasmith, “Biologically plausible, human-scale knowledge representation,” *35th Annual Conference of the Cognitive Science Society*, pp. 412–417, 2013.
- [8] T. Stewart and C. Eliasmith, “Large-scale synthesis of functional spiking neural circuits,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 881–898, 2014, ISSN: 0018-9219.
- [9] T. Sharp and S. Furber, “Correctness and performance of the SpiNNaker architecture,” in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, 2013, pp. 1–8.
- [10] A. Georgopoulos, A. Schwartz, and R. Kettner, “Neuronal population coding of movement direction,” *Science*, vol. 233, no. 4771, pp. 1416–1419, 1986.
- [11] C. Eliasmith, *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.
- [12] T. C. Stewart, Y. Tang, and C. Eliasmith, “A biologically realistic cleanup memory: autoassociation in spiking neurons,” *Cognitive Systems Research*, vol. 12, no. 2, pp. 84–92, Jun. 2011, ISSN: 13890417.
- [13] S. Furber and S. Temple, “Neural systems engineering,” *Journal of The Royal Society Interface*, vol. 4, no. 13, pp. 193–206, 2007, ISSN: 1742-5689.
- [14] T. Bekolay, J. Bergstra, E. Hunsberger, *et al.*, “Nengo: A Python tool for building large-scale functional brain models,” *Frontiers in Neuroinformatics*, vol. 7, no. 48, 2014, ISSN: 1662-5196.
- [15] T. C. Stewart, X. Choo, and C. Eliasmith, “Dynamic behaviour of a spiking model of action selection in the basal ganglia,” in *Proceedings of the 10th international conference on cognitive modeling*, 2010, pp. 235–40.
- [16] F. Galluppi, S. Davies, S. Furber, T. Stewart, and C. Eliasmith, “Real time on-chip implementation of dynamical systems with spiking neurons,” in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, 2012, pp. 1–8.
- [17] E. Stromatias, F. Galluppi, C. Patterson, and S. Furber, “Power analysis of large-scale, real-time neural networks on spinaker,” in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, 2013, pp. 1–8.
- [18] D. F. M. Goodman and R. Brette, “The Brian simulator,” *Frontiers in Neuroscience*, vol. 3, no. 26, 2009, ISSN: 1662-453X.
- [19] N. T. Carnevale and M. L. Hines, *The NEURON book*. Cambridge University Press, 2006.
- [20] M.-O. Gewaltig and M. Diesmann, “Nest (neural simulation tool),” *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.
- [21] G. Wang, “Power analysis and optimizations for gpu architecture using a power simulator,” *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, vol. 1, pp. 619–623, 2010, ISSN: 2154-7491.
- [22] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128 times 128 120 db 15us latency asynchronous temporal contrast vision sensor,” *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 2, pp. 566–576, 2008, ISSN: 0018-9200.
- [23] V. Chan, S.-C. Liu, and A. van Schaik, “Aer ear: A matched silicon cochlea pair with address event representation interface,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, no. 1, pp. 48–59, 2007, ISSN: 1549-8328.
- [24] A. Russell Voelker, E. Crawford, and C. Eliasmith, “Learning large-scale heteroassociative memories in spiking neurons,” in *Unconventional Computation & Natural Computation*, London, Ontario, 2014.