LARGER GPU-ACCELERATED BRAIN SIMULATIONS WITH PROCEDURAL CONNECTIVITY

JAMES C KNIGHT AND THOMAS NOWOTNY

Centre for Computational Neuroscience and Robotics, School of Engineering and Informatics, University of Sussex, Brighton, United Kingdom

Abstract. Simulations are an important tool for investigating brain function but large models are needed to faithfully reproduce the statistics and dynamics of brain activity. Simulating large spiking neural network models has, until now, required so much memory for storing synaptic connections that it could only be done on high performance computer systems. Here, we present an alternative simulation method we call 'procedural connectivity' where connectivity and synaptic weights are generated on the fly' instead of stored and retrieved from memory. This method is particularly well-suited for use on Graphical Processing Units (GPUs)– which are a common fixture in many workstations. Extending our GeNN software with procedural connectivity and a second technical innovation for GPU code generation, we can simulate a recent model of the Macaque visual cortex with 4.13×10^6 neurons and 24.2×10^9 synapses on a single GPU – a significant step forward in making large-scale brain modelling accessible to more researchers.

1. Introduction

The brain of a mouse has around 70×10^6 neurons, but this number is dwarfed by the 1×10^{12} synapses which connect them [1]. In computer simulations of spiking neural networks, propagating spikes involves adding the weighted synaptic input from each spiking presynaptic neuron to the postsynaptic neurons. The information describing which neurons are synaptically connected and with what weight is typically generated before a simulation is run and stored in large arrays. For large-scale brain models this creates high memory requirements, so that they can typically only be simulated on large distributed computer systems using software such as NEST [2] or NEURON [3]. By careful design, these simulators can keep the memory requirements for each node almost constant, even when a simulation is distributed across tens of thousands of nodes [4]. However, high performance computer (HPC) systems are bulky, expensive and consume a lot of power so are typically shared resources, only accessible to a limited number of researchers for time-limited investigations.

Neuromorphic systems [5,6,7,8,9] take inspiration from the brain and have been developed specifically for simulating large spiking neural networks more efficiently. One particular relevant feature of the brain is that its memory elements – the synapses – are co-located with the computing elements – the neurons. In neuromorphic systems, this often translates to dedicating a large proportion of each chip to memory. This allows some classes of spiking neural networks to be simulated very efficiently but, reducing the degree of connectivity to fit within the constraints of current neuromorphic systems inevitably changes the dynamics of brain simulations [10]. However, while such on-chip memory is fast, it can only be fabricated at relatively low density so that many of these systems economize – either by reducing the maximum number of synapses per neuron to as few as 256 or by reducing the precision of the synaptic weights to 6 bit [9], 4 bit [5] or even 1 bit [7]. Unlike most other neuromorphic systems, the SpiNNaker [6] neuromorphic supercomputer is fully programmable and combines large on-chip and external memories, distributed across the system, which enables real-time simulation of large-scale models [11]. This is promising for the future but, due to its prototype nature, the availability of SpiNNaker hardware is

E-mail address: J.C.Knight@sussex.ac.uk.

limited and even moderately-sized simulations still require a physically large system (9 boards for a model with around 100×10^3 neurons and 300×10^6 synapses[11]).

Modern GPUs have relatively little on-chip memory and, instead, dedicate the majority of their silicon area to arithmetic logic units. GPUs use dedicated hardware to rapidly switch between tasks so that the latency of accessing external memory can be 'hidden' behind computation, as long as there is sufficient computation to be performed. For example, the memory latency of a typical modern GPU can be completely hidden if each CUDA core performs approximately 10 arithmetic operations per byte of data accessed from memory. Unfortunately, propagating a spike in a spiking neural network simulation typically requires reading a synaptic weight ($\geq 2\,\mathrm{B}$) and postsynaptic index ($\geq 2\,\mathrm{B}$) from memory and then writing back the synaptic input ($\geq 4\,\mathrm{B}$), while performing many fewer than the corresponding 80 instructions. This makes spike propagation highly memory bound. Nonetheless, we have shown in previous work[12] that, as GPUs have significantly higher total memory bandwidth than even the fastest CPU, moderately sized models of around 100×10^3 neurons and 1×10^9 synapses can be simulated on a single GPU with competitive speed and energy consumption. However, individual GPUs do not have enough memory to simulate larger brain models. One can install a small number of GPUs within a single node and connect them using high-speed NVLink[13] interconnect, through which they can communicate with low latency and high bandwidth[14]. However, if a larger number of GPUs is required, multiple nodes need to be used which are connected via the same MPI based protocols and network fabric as CPU based systems and hence suffer from similar communication overheads as CPU based clusters.

In this work, we present two innovations which enable large-scale brain simulations on a single GPU workstation. The first innovation, which we describe in section 2.1, we call 'procedural connectivity'. Procedural connectivity uses the large amount of computational power available on a GPU to generate synaptic connectivity 'on the fly' – reducing memory and memory bandwidth requirements. While a similar approach was used by Eugene Izhikevich for simulating an extremely large thalamo-cortical model with 1×10^{11} neurons and 1×10^{15} synapses on a modest PC cluster in 2005[15] – an incredible achievement – it has not been subsequently applied to modern hardware. Procedural connectivity vastly reduces the memory requirements of large-scale brain simulations but, previous GPU code generation strategies do not scale well to models containing large numbers of neural populations and synaptic projections. To address this second problem, we have developed a 'kernel merging' code generator, described in section 2.2. In section 2.3 we then demonstrate the power of these new features by simulating a recent model of the Macaque visual cortex[16] with 4.13×10^6 neurons and 24.2×10^9 synapses.

2. Results

2.1. **Procedural connectivity.**

In a brain simulation, neurons and synapses can be described by a variety of mathematical models but these are eventually all translated into time or event-driven update algorithms[17]. Our GeNN simulator[18] uses code generation to convert neuron and synapse update algorithms – described using 'snippets' of C-like code – into CUDA code for efficient GPU simulation. Before a simulation can be run, its parameters, in particular the state variables and the synaptic connectivity, need to be initialised. Traditionally, this is done by running initialisation algorithms on the main CPU prior to the simulation. The results are stored in CPU memory, uploaded to GPU memory and then used during the simulation. We have recently extended GeNN to use code generation from code snippets to also generate efficient, parallel code for model initialisation[12]. Offloading initialisation to the GPU in this way made it around 20× faster on a desktop PC[12], demonstrating that initialisation algorithms are well-suited for GPU acceleration. Here, we are going one step further. We realised that, if each synaptic connection can be re-initialised in less than the 80 operations required to hide the latency incurred when fetching its 8 B of state from memory, it could be faster and vastly more memory efficient to regenerate synaptic connections on demand rather than storing them in memory. This is the concept of procedural connectivity. It is applicable whenever synapses are static – plastic synapses which change their weights during a simulation will have to be simulated in the traditional way.

We implemented procedural connectivity in GeNN by repurposing our previously developed parallel initialisation methods. Instead of running them once for all synapses at the beginning of the simulation, we rerun the methods during the simulation to regenerate the outgoing synapses of each neuron that fires a spike and immediately use the generated connections and weights to run the post-synaptic code which calculates the effect of the spike onto other neurons. This is possible because the outgoing synaptic connections from each neuron are typically largely independent from those of other neurons as we shall see from typical examples below.

In the absence of knowledge of the exact microscopic connectivity in the brain, there are a number of typical connectivity schemes that are used in brain models. We will now discuss two typical examples and how they can be implemented efficiently on a GPU. One very common connectivity scheme is the 'fixed probability connector' for which each neuron in the presynaptic population is connected to each neuron in the postsynaptic population with fixed probability P_{conn}. The postsynaptic targets of any presynaptic neuron can hence be sampled from a Bernoulli process with success probability P_{conn}. One simple way of sampling from the Bernoulli process is to repeatedly draw samples from the uniform distribution Unif$[0, 1]$ and generate a synapse if the sample is less than P_{conn}. However, for sparse connectivity ($P_{\text{conn}} \ll 1$), it is much more efficient to sample from the geometric distribution Geom$[P_{\text{conn}}]$ which governs the number of Bernoulli trials until the next success (i.e. a synapse). The geometric distribution can be sampled in constant time by inverting the cumulative density function of the equivalent continuous distribution (the exponential distribution) to obtain $\frac{log(\text{Unif}[0,1])}{log(1-P_{\text{conn}})}$ [19] p499. Note that, if we were to directly draw from the uniform distribution, the sampling for each potential synapse would be independent from any other potential synapse and all these operations could be performed in parallel. However, for the more efficient 'geometric sampling' employed here, the sampling for the post-synaptic targets of a presynaptic neuron must be done serially, but is still independent from the sampling for any other presynaptic neuron.

Another common scheme for defining connectivity is the 'fixed total number connector' in which a fixed total number N_{syn} of synapses is placed between randomly chosen partners from the pre- and postsynaptic populations. In order to initialise this connectivity in parallel, the number of synapses that originate from each of the N_{pre} presynaptic neurons must first be calculated by sampling from the multinomial distribution Mult$[N_{\text{syn}}, \{\frac{1}{N_{\text{pre}}}, \frac{1}{N_{\text{pre}}}, \ldots, \frac{1}{N_{\text{pre}}}\}]$ up front on the host CPU because these numbers need to add to N_{syn} and are hence not independent. However, once the numbers of outgoing synapses are determined, the postsynaptic targets for a presynaptic neuron can be generated very efficiently in parallel by sampling from the discrete uniform distribution Unif$[0, N_{\text{post}}]$ where N_{post} is the size of the postsynaptic population. Note, that this can only be done because the targets of each presynaptic neuron are independent from those of any other presynaptic neuron. Where synaptic weights and delays are not constant across synapses, but are described by some statistical distribution, they can also be sampled independently from each other and hence in parallel.

In order to use these parallel initialisation schemes for procedural connectivity, we require reproducible pseudorandom numbers that can be generated independently for each presynaptic neuron. In principle this could be done with 'conventional' pseudorandom number generators (PRNGs), but each presynaptic neuron would need to maintain its own PRNG state which would lead to a significant memory overhead. Instead, we use the 'counter-based' Philox4×32-10 PRNG [20]. Counter-based PRNGs are designed for parallel applications and essentially consist of a pseudo-random bijective function which takes a counter as an input (for Philox4×32-10 a 128 bit number) and outputs a random number. In contrast to convential PRNGs, this means that generating the n^{th} random number in a stream has exactly the same cost as generating the 'next' random number, allowing us to trivially divide up the random number stream between multiple parallel processes (in this case presynaptic neurons).

For an initial demonstration of the performance and scalability of procedural connectivity, we simulated a balanced random network of Leaky Integrate-and-Fire (LIF) neurons (see section 4.2 for model description) at scales ranging from 1×10^3 to 1×10^6 neurons (100×10^3 to 100×10^9 synapses respectively) on a representative selection of NVIDIA

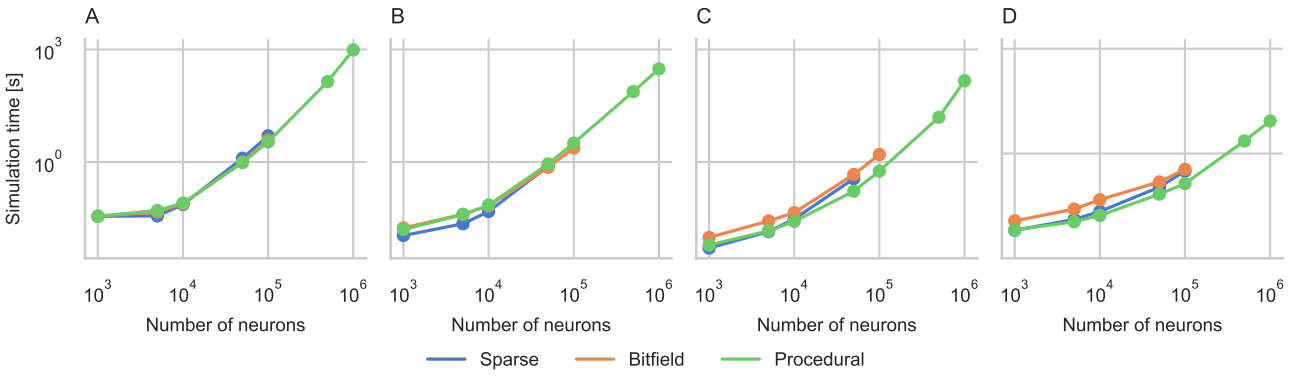

FIGURE 1. Scaling of 1 s balanced random network simulation using different algorithms on a range of modern GPUs. All data points represent the mean of 5 simulations and standard deviations are smaller than line width so not shown. **A** Jetson TX2. **B** GeForce MX130. **C** GeForce GTX 1650. **D** Titan RTX.

GPU hardware: Jetson TX2, a low-power embedded system with 8 GB (shared memory); Geforce MX130, a laptop GPU with 2 GB; Geforce GTX 1650, a low-end desktop GPU with 4 GB; and Titan RTX, a high-end workstation GPU with 24 GB. Fig. 1 shows the duration of these simulations using our new procedural approach and the standard approach of storing synaptic connections in memory employing two different data structures. Both data structures are described in more detail in our previous work [12] but briefly, in the 'sparse' data structure, a presynaptic neuron's postsynaptic targets are represented as an array of indices whereas, in the 'bitfield' data structure, they are represented as a N_{post} array of bits where a '1' at position i indicates that there is a connection to postsynaptic neuron i and a '0' that there is not. In order to verify our procedural connectivity algorithm and its implementation, we also recorded membrane voltages from two instantiations of the model with fixed random seeds and both sparse and procedural connectivity. We found that, to 32 bit floating point precision, the membrane voltages of all neurons were identical.

None of our devices have enough memory to store the 100×10^9 synapses required for the largest scale using either data structure but, at the 100×10^3 neuron scale, the bitfield data structure allows the model to fit into the memory of several devices it otherwise would not. However, not only is the new procedural approach the *only* way of simulating models at the largest scales but, as Fig. 1 illustrates, even at smaller scales the performance of the procedural approach is competitive with and sometimes better than the standard approach. Having removed the reading of synaptic weights and postsynaptic indices from memory, one might expect even higher performance. However, our current implementation still accumulates postsynaptic input using a 'fire-and-forget' atomic add operation. Atomic operations are entirely handled within the L2 cache of modern GPUs and thus have relatively low latency. However, profiling using NVIDIA Nsight compute [21] suggests that the latency of these operations, combined with those incurred when sampling from the PRNG and calculating logarithms are currently the limiting factor in this benchmark rather than the compute resources of the GPU, suggesting that further optimisation could significantly improve performance. All of the synapses in this model have the same synaptic weight meaning that they can be hard-coded into the procedural connectivity kernels. However, if weights vary across synapses, the 'bitfield' cannot be used and the memory constraints for the 'sparse' representation become even more severe.

2.2. **Kernel merging.** NVIDIA GPUs are typically programmed in CUDA using a Single Instruction Multiple Thread (SIMT) paradigm where programmers write 'kernel' functions containing serial C-like code which is then executed in parallel across many virtual threads. We call our second innovation 'kernel merging' and it relates to the way these kernels are implemented. While the procedural connectivity presented in the previous section allows models which would not otherwise fit into the memory of a GPU to be simulated, there are

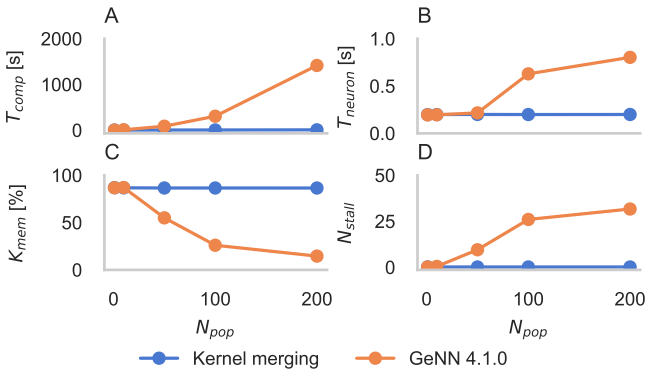

FIGURE 2. Performance of a 1 s simulation of 1×10^6 LIF neurons driven by a Gaussian input current, partitioned into varying numbers (N_{pop}) of populations and running on a workstation equipped with a Titan RTX GPU. All data points represent the mean of 5 simulations and standard deviations are smaller than line width so not shown. **A** Compilation time (T_{comp}) using GCC 7.5.0. **B** Neuron kernel time (T_{neuron}) for a 1 s simulation. **C** Memory throughput (K_{mem}) reported by NVIDIA Nsight compute profiler "Speed of light" metric. **D** Number of "No instruction" stalls reported by NVIDIA Nsight compute profiler (N_{stall}).

additional problems when using code generation for models with a large number of neuron and synapse populations. GeNN and other SNN simulators which use code generation to generate all of their simulation code[22] (as opposed to, for example NESTML[23], which uses code generation only to generate neuron simulation code) generate separate pieces of code for each population of neurons and synapses. This allows optimizations such as hard-coding constant parameters and, although generating code for models with many populations will result in large code size, C++ CPU code can easily be divided between multiple modules and compiled in parallel, minimizing the effects on build time. However, GPUs can only run a small number of kernels – which are equivalent to modules in this context – simultaneously (128 on the latest NVIDIA GPUs[24] p278). Therefore, GeNN employs a "Kernel Fusion" approach[25] where multiple neuron populations are simulated within each kernel. CUDA threads are grouped into 'thread blocks' and to maximise performance, the code running on threads within a thread block should not 'diverge'. To minimise this divergence, we add 'padding' threads around each population of neurons so population and block boundaries are aligned. Therefore, in the following pseudocode we simulating 3 populations of 100 neurons each in a single kernel and – assuming a thread block size of 32 – we pad each population to 128 threads:

```
void updateNeurons() {
  if(thread < 128) {
    if(thread < 100) {
      // Update neuron population A
    }
  } else if(thread >= 128 && thread < 256) {
    if((thread - 128) < 100) {
      // Update neuron population B
    }
  } else if(thread >= 256 && thread < 384) {
    if((thread - 256) < 100) {
      // Update neuron population C
    }
  }
```

```
}
```

This approach works well for a small number of populations but, as Fig. 2A illustrates, when we partition a model consisting of a single population of 1 000 000 LIF neurons (see section 4.3 for model description) into an increasingly large number of ever smaller populations, compilation time increases super-linearly and quickly becomes impractical. Furthermore, the simulation also runs slower with a large number of populations (Fig. 2B).

Normally, we would expect this model to be memory bound as each thread in the model reads 32 B of data (8 B of neuron state and 24 B of RNG state) and, as discussed above, hiding the latency of these memory accesses would require approximately 320 arithmetic operations – many more than required to sample an input current from the normal distribution and update a LIF neuron. Fig. 2C – obtained using data from the NVIDIA Nsight compute profiler[21] – shows that this is true for small numbers of populations. In this case, the memory system is around 90 % utilised. However, when the model is partitioned into larger numbers of smaller populations, the memory is used less efficiently and the kernel becomes latency bound, i.e. neither memory *nor* compute are used efficiently. Investigating further, we found that this drop in performance was accompanied by an increasing number of "No instruction" stalls (Fig. 2D) which are events that prevent the GPU from doing any work during a clock cycle. These particular events are likely to be caused by "Excessively jumping across large blocks of assembly code" [21] p47, which makes sense as we are generating kernels with hundreds of thousands of lines of code.

Several neural modelling tools including Brian2[26] provide modellers with tools to work with 'slices' of neuron populations, allowing models to be defined with fewer populations. However, if a model is defined by connecting these slices together, the resulting connectivity is the result of *multiple* simple connection rules of the type discussed in the previous section, making it much more difficult to apply our procedural connectivity approach. Furthermore, such an approach places the responsibility for structuring a model in such a way that it can be simulated efficiently onto the modellers, who often prefer to concentrate on the science and organise populations according to anatomy or physiology.

To address the issue of too many populations, we developed a new code generator for GeNN which applies a Single Program Multiple Data (SPMD) approach and 'merges' the model description, grouping together populations which can be simulated using the same generated code. From this merged description, structures are generated to store the pointers to state variables and parameter values which are still allowed to differ between merged populations:

```
struct NeuronUpdateGroup {
  unsigned int numNeurons;
  float* V;
};
```

An array of these structures is then declared for each merged population and each element is initialised with pointers to state variables and parameter values:

```
NeuronUpdateGroup neuronUpdateGroup[3];
neuronUpdateGroup[0] = {100, VA};
neuronUpdateGroup[1] = {100, VB};
neuronUpdateGroup[2] = {100, VC};
```

where VA is a pointer to the array containing the state variable 'V' of populations 'A' and so on. In order for a thread to determine which neuron in which population it should simulate, we generate an additional data structure – an array containing a cumulative sum of threads used for each population:

```
unsigned int startThread[3] = {0, 128, 256};
```

Each thread performs a simple binary search within this array to find the index of the neuron and population it should simulate. As Fig. 2 shows, this approach solves the observed issues with compilation time and simulation performance. Interestingly, the additional overhead of a binary search does not affect the performance even for a large number of populations as the kernel execution is generally memory bound.

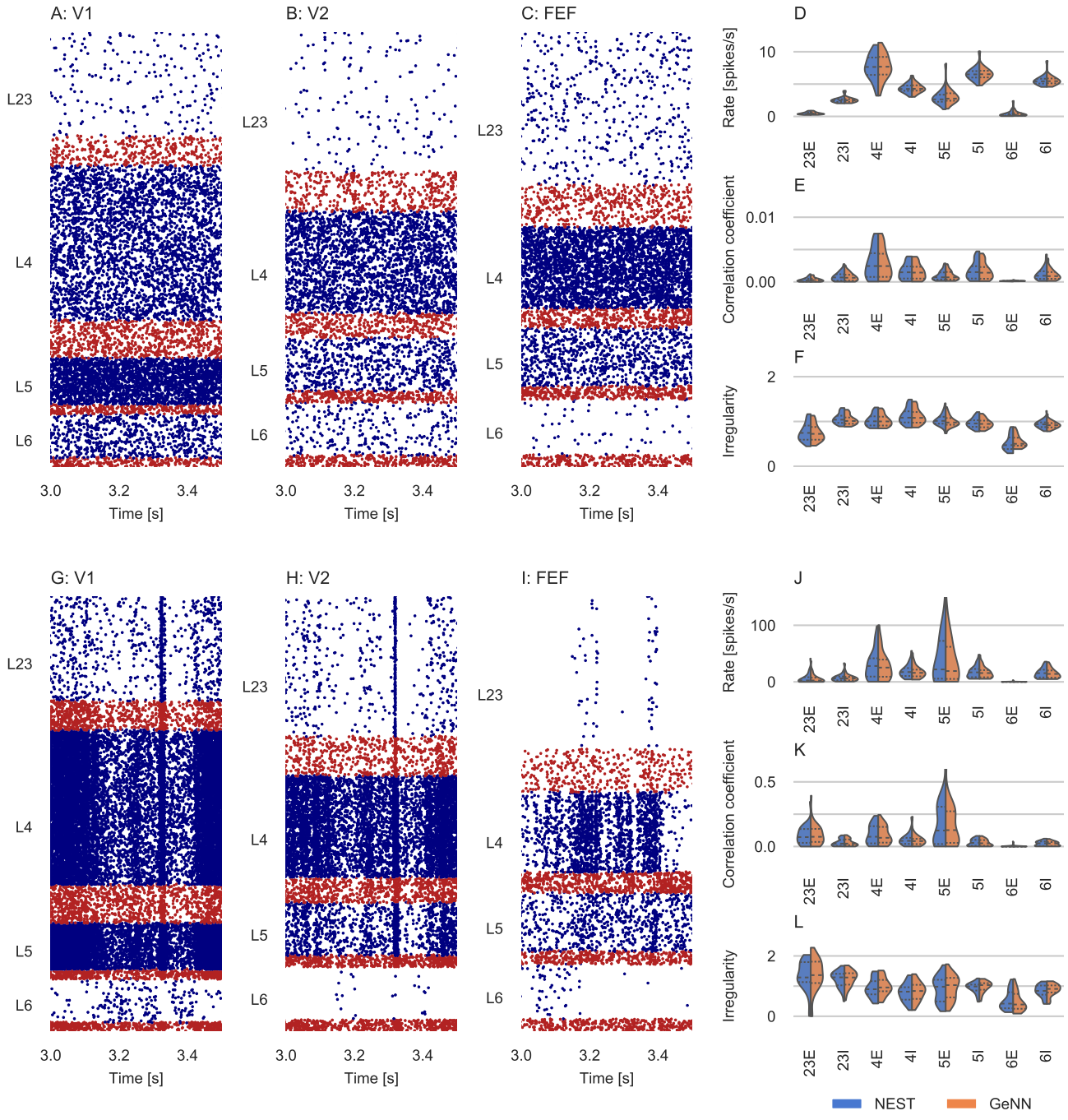

FIGURE 3. Results of full-scale multi-area model simulation. **A-C** Raster plots from ground state simulation in GeNN, showing spiking activity of 3 % of the neurons in area V1 (A), V2 (B), and the Frontal Eye Field (FEF) (C). Blue: excitatory neurons, red: inhibitory neurons. **D-F** Spiking statistics across all 32 areas for each population for ground state simulations, using GeNN (orange) and NEST (blue), shown as split violin plots. Population-averaged firing rates (D), Average pairwise correlation coefficients of spiking activity (E) and Irregularity measured by revised local variation LvR[27] averaged across neurons (F). Solid lines in violins: medians, Dashed lines: Interquartile range. **G-I** Raster plots as in (A-C) but for resting state simulation in GeNN. **J-L** Same spiking statistics as (D-F) but for resting state simulation.

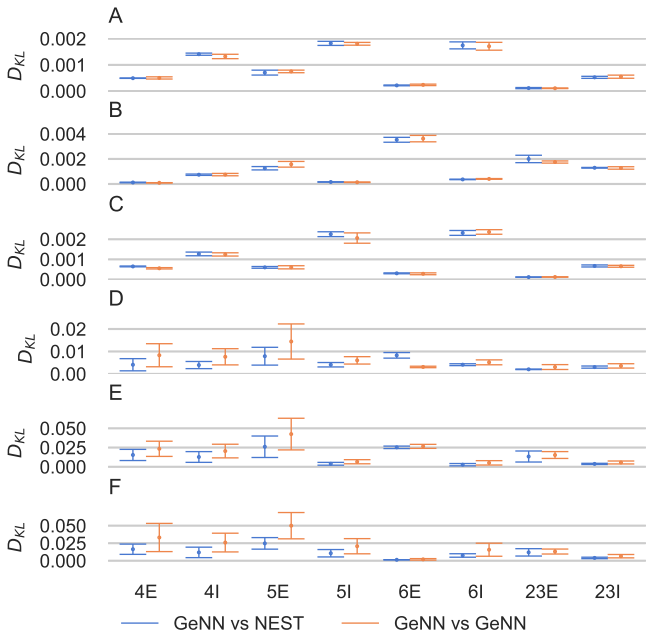

FIGURE 4. Comparison of full-scale multi-area model spike statistics between multiple GeNN simulations with different seeds; and between GeNN and NEST simulations. Comparisons calculated as the Kullback-Leibler (KL) divergence between the types of distribution displayed in Fig. 3D-F,J-L. Bullets mark the mean and whiskers indicate the standard deviation of observed KL divergences across different pairs of compared distributions. **A,B** Neuron firing rates for ground state (A) and resting state B) **C,D** Pairwise correlation coefficients of spiking activity. **E,F** Irregularity measured by revised local variation LvR [27].

2.3. **The multi-area model.** Due to lack of computing power and sufficiently detailed connectivity data, previous models of the cortex have either focused on modelling individual local microcircuits at the level of individual cells [28,29] or modelling multiple connected areas at a higher level of abstraction [30]. However, recent data [31] has shown that cortical activity has distinct features at both the global and local levels which can only be captured by modelling interconnected microcircuits at the level of individual cells. The recent multi-area model [32,16] is an example of such multi-scale modeling. It uses scaled versions of a previous, 4 layer microcircuit model [29] to implement 1 mm^2 'patches' for 32 areas of the macaque visual cortex. The patches are connected together according to inter-area axon tracing data from the CoCoMac [33] database, further refined using additional anatomical data [34] and heuristics [35] to obtain estimates for the number of synapses between areas. The synapses are distributed between populations in the source and target area using layer-specific tracing data [36] and cell-type-specific dendritic densities [37] (see section 4.4 for model description). In 2018, this model was simulated using NEST [2] on an IBM Blue Gene/Q supercomputer. Because each Blue Gene/Q compute node only had a small amount of memory, these simulation were distributed across an entire rack (1024 compute nodes) of the system and simulating 1 s of biological time took approximately 12 min [16]. However, on the newer 'JURECA' supercomputer (where each node has 128 GB of memory) this model can be simulated on as few as 11 compute nodes. Furthermore, by distributing the model across 160 compute nodes, it can be initialized in approximately 8 min and simulating 1 s of biological time takes only 31 s [38].

The multi-area model consists of 4.13×10^6 neurons in 254 populations and 24.2×10^9 synapses in 64 516 projections. Without kernel merging, it would therefore be unlikely that the model would compile or simulate at a workable speed using GeNN. Additionally,

unlike the model we benchmarked previously, each synapse in this model has an independent weight and synaptic delay sampled from a normal distribution so the bitfield data structure cannot be used. Even if we assume that 16 bit floating-point would provide sufficient weight precision, that delays could be expressed as 8 bit integers and that neuron populations are all small enough to be indexed using 16 bit indices, our sparse data structure would still require 5 B per synapse, such that the complete synaptic data would need over 100 GB of GPU memory. While a cluster of GPUs connected using NVLink could be built with this much memory, it is more than any single GPU has available. However, using procedural connectivity, we are able to simulate this model on a single workstation with a Titan RTX GPU.

In order to test our implementation of the multi-area model in GeNN, we ran a 10.5 s simulation of the model in a 'ground state' where inter-area connections have the same strength as intra-area connections and a 100.5 s simulation in a 'resting state' where inter-area connections are 1.9× stronger. Initialization of our model took 6 min (3 min of which was spent generating and compiling code) and simulation of each biological second took 7.9 min in the ground state and 8.5 min in the resting state (averaged over 3 simulation runs). While this is significantly faster than the older Blue Gene/Q supercomputer, it is around 15× slower than the newer JURECA system. However, because individual synapses simulated using procedural connectivity do not require initialization, in shorter simulations this large performance difference can be somewhat offset by the faster initialisation time of the GeNN simulations. For example a 1 s simulation would take just under 11 min using GeNN (if the model is not recompiled) and around 8.5 min using 160 JURECA nodes – only 2.5 min faster.

Fig. 3A-C shows some example spike rasters from three of the modelled areas, illustrating the asynchronous irregular nature of the model's ground state whereas Fig. 3G-I illustrate the characteristic irregular activity and population bursts of the same areas in the resting state. Next, we calculated the per-layer distributions of rates, spike-train irregularity and cross-correlation coefficients across all areas (disregarding the first 500 ms of simulation) and compared them to the same measures obtained from spike trains generated by the supercomputer simulations. We calculated irregularity using the revised local variation LvR [27], averaged over a subsample of 2000 neurons and cross-correlation from spike histograms with 1 ms bins, calculated from a subset of 2000 non-silent neurons. The distributions of these values – obtained from the NEST and GeNN simulations – are shown as violin plots in Fig. 3D-F and Fig. 3J-L. Upon visual inspection, the distributions are very similar but, to assess how meaningful the differences between these distributions are, we compared the distributions obtained from 3 GeNN simulations with different random number generator seeds to those obtained from NEST simulations. We performed this comparison by computing histograms of the three measures (using bin sizes determined with the Freedman-Diaconis rule [39]) and comparing pairs of distributions using the Kullback-Leibler divergence D_{KL}. The results of these comparisons are shown in Fig. 4 and suggest that the influence of the random seeds is comparable to the influence of random seeds and simulator combined – indicating that the choice of simulator has little effect on the model dynamics.

3. Discussion

In this work we have presented a novel approach for large-scale brain simulation on GPU devices which entirely removes the need to store connectivity data in memory. We have shown that this approach allows us to simulate a cortical model with 4.13×10^6 neurons and 24.2×10^9 synapses [32,16] on a single modern GPU. While this represents a significant step forward in terms of making large-scale brain modelling tools accessible to a large community of brain researchers, this model still has around 20× fewer neurons and 40× fewer synapses than the brain of even a small mammal such as a mouse [1]. Our implementation of the multi-area model requires a little over 12 GB of GPU memory, with the majority (8.5 GB) being used for the circular dendritic delay buffers (see Knight and Nowotny [12]). These are a per-neuron (rather than per-synapse) data structure but, because the inter-area connections in the model have delays of up to 500 simulation timesteps, the delay buffers become quite large. However, for models without large delays such as

Parameter	Procedural connectivity benchmark	Merging benchmark	Multi-area model
τ_m [ms]	20	20	10
V_rest [mV]	−60.0	−70.0	−65
V_th [mV]	−50.0	−51.0	−50
R_m [MΩ]	20	20	40
τ_ref [ms]	5	2	2

TABLE 1. Neuron parameters.

the balanced random network simulated in section 2.1, only around 20 B is required per neuron meaning that, theoretically, over 1×10^9 neurons (over 10× more than a mouse brain) could be simulated on a 24 GB GPU.

One additional benefit of procedural connectivity not explored in this work is that parameters such as connection density or weight variance can be changed at runtime with minimal overhead. Furthermore, these modifications could be driven by the state of the network to implement a primitive form of structural plasticity. More generally, synaptic plasticity and learning are both important aspects of large-scale brain simulation not addressed in this work. As discussed by Knight and Nowotny [12], GeNN supports a wide variety of synaptic plasticity rules. In order to modify synaptic weights, they need to be stored in memory rather than generated procedurally. However, connectivity could still be generated procedurally, potentially halving the memory requirements of models with synaptic plasticity. This would be sufficient for synaptic plasticity rules that only require access to presynaptic spikes and postsynaptic neuron states [40,41] but, for many Spike-Timing-Dependent Plasticity (STDP) rules, access to *postsynaptic* spikes is also required. GeNN supports such rules by automatically generating a lookup table structure (see Knight and Nowotny [12]). While this process could be adapted to generate a lookup table from procedural connectivity, this would further erode memory savings. However, typically not all synapses in a simulation are plastic and those that are not could be simulated fully procedurally.

In this work, we have discussed the idea of procedural connectivity in the context of GPU hardware but, we believe that there is also potential for developing new types of neuromorphic hardware built from the ground up for procedural connectivity. The latencies of sampling from the random number generator and performing atomic add operations were identified in section 2.1 as the factors currently limiting performance of the procedural connectivity algorithm. By dedicating more silicon area to fast on-chip memory – which would alleviate the need for atomic operations – and by implementing the random number generator in hardware, these limitations could be overcome, leading to truly game-changing compute time improvements.

4. METHODS

4.1. **Neuron model.** In all experiments presented in this work, neurons are modelled as Leaky Integrate-and-Fire (LIF) units with the parameters listed in Table 1. The membrane voltage V_i of neuron i is modelled as

$$\tau_\mathrm{m} \frac{dV_i}{dt} = (V_\mathrm{rest} - V_i) + R_\mathrm{m}(I_{\mathrm{syn}_i} + I_{\mathrm{ext}_i}), \tag{1}$$

where τ_m and R_m represent the time constant and resistance of the neuron's cell membrane, V_rest defines the resting potential, I_{syn_i} represents the synaptic input current and I_{ext_i} represents an external input current. When the membrane voltage crosses a threshold V_th a spike is emitted, the membrane voltage is reset to V_rest and updating of V is suspended for a refractory period τ_ref. In the models where there are synaptic connections, they are current-based, i.e. pre-synaptic spikes lead to exponentially-decaying input currents I_{syn_i}

$$\tau_\mathrm{syn} \frac{dI_{\mathrm{syn}_i}}{dt} = - I_{\mathrm{syn}_i} + \sum_{i=0}^{n} w_{ij} \sum_{t_j} \delta(t - t_j), \tag{2}$$

where τ_{syn} represents the decay time constant and t_j are the arrival times of incoming spikes from n presynaptic neurons. The ordinary differential equations (1) and (2) are solved with an exponential Euler algorithm.

4.2. **Balanced random network model.** This model was first presented by Vogels and Abbott [42] but has subsequently been widely used as a scalable benchmark [17]. The network consists of N LIF neurons, modelled using the approach described in section 4.1 and configured with the parameters shown in table 1. The neurons are partitioned into one population of $\frac{4N}{5}$ excitatory and a second of $\frac{N}{5}$ inhibitory neurons. The two populations of neurons are connected to each other and with themselves with fixed probability $P_{\text{conn}} = 10\,\%$ (the highest density at which Vogels and Abbott [42] suggests their results hold). All excitatory synapses have $\tau_{\text{syn}} = 5\,\text{ms}$ and $w_{ij} = \frac{3.2}{N}\,\text{nA}$; and all inhibitory synapses have $\tau_{\text{syn}} = 10\,\text{ms}$ and $w_{ij} = \frac{40.8}{N}\,\text{nA}$. Additionally, every cell is depolarized by approximately 10 mV by applying a constant external current $I_{\text{ext}_j} = 0.55\,\text{nA}$. Simulations were run with a time step $\Delta t = 1.0\,\text{ms}$.

4.3. **Merging model.** This model simply consists of 1 000 000 LIF neurons, again modelled using the approach described in section 4.1 and configured with the parameters shown in table 1. Each neuron is driven by an independent Gaussian input current of $I_{\text{ext}_j} \sim \mathcal{N}(1.0, 0.25^2)$. Simulations were run with a time step $\Delta t = 1.0\,\text{ms}$.

4.4. **Multi-area model.** For a full description of the model, please refer to tables 2 and 3 in Schmidt et al. [16]. In our implementation, neurons are modelled using the approach described in section 4.1 and configured with the parameters shown in table 1. Background Poisson input is delivered to each neuron via I_{ext_i} with

$$\tau_{\text{ext}} \frac{dI_{\text{ext}_i}}{dt} = -I_{\text{ext}_i} + \text{Poisson}(\frac{\nu_{\text{ext}} \Delta t}{1000}), \tag{3}$$

where $\tau_{\text{syn}} = 0.5\,\text{ms}$ and ν_{ext} is calculated as described in table 2 of Schmidt et al. [16]. Individual populations are connected by the fixed number connectors described in section 2.1. Simulations were run with a time step $\Delta t = 0.1\,\text{ms}$.

5. Acknowledgments

We would like to thank Jari Pronold, Sacha van Albada, Agnes Korcsak-Gorzo and Maximilian Schmidt for their help with the multi-area model data; and Dan Goodman and Mantas Mikaitis for their feedback on the manuscript. This work was funded by the EPSRC (Brains on Board project, grant number EP/P006094/1).

6. Author contributions

J.K. and T.N. wrote the paper. T.N. is the original developer of GeNN. J.K. is currently the primary GeNN developer and was responsible for extending the code generation approach to the procedural simulation of synaptic connectivity. J.K. performed the experiments and the analysis of the results that are presented in this work.

7. Data availability

The raw data used to produce Fig. 1 and Fig. 2; and the pre-processed data used to produce Fig. 3 are available at `https://github.com/BrainsOnBoard/procedural_paper`. The raw spiking data from the GeNN simulations of the multi-area model are available at `https://doi.org/10.25377/sussex.12912699`. The raw spiking data from the NEST simulations run by Schmidt et al. [16] are available from the original authors upon request.

8. Code availability

All experiments were carried out using the GeNN 4.3.3, available at `https://github.com/genn-team/genn/releases/tag/4.3.3`. A GeNN port of the multi-area model is available at `https://github.com/neworderofjamie/multi-area-model`. The models used to produce Fig. 1 and Fig. 2 are all available at `https://github.com/BrainsOnBoard/procedural_paper`.

References

[1] S. Herculano-Houzel, B. Mota, and R. Lent. Cellular scaling rules for rodent brains. *Proceedings of the National Academy of Sciences*, 103(32):12138–12143, aug 2006. ISSN 0027-8424. doi: 10.1073/pnas.0604911103. URL http://www.pnas.org/cgi/doi/10.1073/pnas.0604911103.

[2] Marc-Oliver Gewaltig and Markus Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007.

[3] Nicholas T Carnevale and Michael L Hines. *The NEURON book*. Cambridge University Press, 2006.

[4] Jakob Jordan, Tammo Ippen, Moritz Helias, Itaru Kitayama, Mitsuhisa Sato, Jun Igarashi, Markus Diesmann, and Susanne Kunkel. Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. *Frontiers in Neuroinformatics*, 12:2, 2018. ISSN 1662-5196. doi: 10.3389/fninf.2018.00002. URL http://journal.frontiersin.org/article/10.3389/fninf.2018.00002/full.

[5] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28nm CMOS. *IEEE Transactions on Biomedical Circuits and Systems*, 13(1):145–158, 2019. ISSN 1932-4545. doi: 10.1109/TBCAS.2018.2880425. URL http://arxiv.org/abs/1804.07858https://ieeexplore.ieee.org/document/8528875/.

[6] Stephen B Furber, Francesco Galluppi, S Temple, and Luis A Plana. The SpiNNaker Project. *Proceedings of the IEEE*, 102(5):652–665, may 2014. ISSN 0018-9219. doi: 10.1109/JPROC.2014.2304638.

[7] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D Flickner, William P Risk, Rajit Manohar, and Dharmendra S Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. doi: 10.1126/science.1254642. URL http://www.sciencemag.org/content/345/6197/668.abstract.

[8] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience*, 9:1–17, 2015. ISSN 1662453X. doi: 10.3389/fnins.2015.00141.

[9] Johannes Schemmel, Laura Kriener, Paul Müller, and Karlheinz Meier. An accelerated analog neuromorphic hardware system emulating NMDA- and calcium-based non-linear dendrites. *Proceedings of the International Joint Conference on Neural Networks*, pages 2217–2226, 2017. doi: 10.1109/IJCNN.2017.7966124.

[10] Sacha Jennifer van Albada, Moritz Helias, and Markus Diesmann. Scalability of Asynchronous Networks Is Limited by One-to-One Mapping between Effective Connectivity and Correlations. *PLoS Computational Biology*, 11(9):1–37, 2015. ISSN 15537358. doi: 10.1371/journal.pcbi.1004490.

[11] Oliver Rhodes, Luca Peres, Andrew G. D. Rowley, Andrew Gait, Luis A. Plana, Christian Brenninkmeijer, and Steve B. Furber. Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164):20190160, feb 2020. ISSN 1364-503X. doi: 10.1098/rsta.2019.0160. URL https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0160.

[12] James C. Knight and Thomas Nowotny. GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model. *Frontiers in Neuroscience*, 12:1–19, 2018. ISSN 1662-453X. doi: 10.3389/fnins.2018.00941. URL https://www.frontiersin.org/article/10.3389/fnins.2018.00941/full.

[13] NVIDIA Corporation. *NVLink Fabric Multi-GPU Processing*, 2020. URL https://www.nvidia.com/en-gb/data-center/nvlink/.

[14] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):94–110, 2020. ISSN 15582183. doi: 10.1109/TPDS.2019.2928289.

[15] Eugene M Izhikevich. Large-Scale Simulation of the Human Brain, 2005. URL `http://www.izhikevich.org/human_brain_simulation/Blue_Brain.htm`.

[16] Maximilian Schmidt, Rembrandt Bakker, Kelly Shen, Gleb Bezgin, Markus Diesmann, and Sacha Jennifer van Albada. A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. *PLoS Computational Biology*, 14(10):1–38, 2018. ISSN 15537358. doi: 10.1371/journal.pcbi.1006359.

[17] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–98, dec 2007. ISSN 0929-5313. doi: 10.1007/s10827-007-0038-6. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2638500/`.

[18] Esin Yavuz, James Turner, and Thomas Nowotny. GeNN: a code generation framework for accelerated brain simulations. *Scientific Reports*, 6:18854, 2016. ISSN 2045-2322. doi: 10.1038/srep18854. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4703976`.

[19] Luc Devroye. *Non-uniform random variate generation*. Springer-Verlag New York, New York, 2013. ISBN 9781461386452.

[20] John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. Parallel random numbers: As Easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, volume 81, page 1, New York, New York, USA, 2011. ACM Press. ISBN 9781450307710. doi: 10.1145/2063384.2063405. URL `http://dl.acm.org/citation.cfm?doid=2063384.2063405`.

[21] NVIDIA Corporation. *Nsight Compute*, 2020. URL `https://docs.nvidia.com/nsight-compute/pdf/NsightCompute.pdf`.

[22] Inga Blundell, Romain Brette, Thomas A. Cleland, Thomas G. Close, Daniel Coca, Andrew P. Davison, Sandra Diaz-Pier, Carlos Fernandez Musoles, Padraig Gleeson, Dan F. M. Goodman, Michael Hines, Michael W. Hopkins, Pramod Kumbhar, David R. Lester, Bóris Marin, Abigail Morrison, Eric Müller, Thomas Nowotny, Alexander Peyser, Dimitri Plotnikov, Paul Richmond, Andrew Rowley, Bernhard Rumpe, Marcel Stimberg, Alan B. Stokes, Adam Tomkins, Guido Trensch, Marmaduke Woodman, and Jochen Martin Eppler. Code Generation in Computational Neuroscience: A Review of Tools and Techniques. *Frontiers in Neuroinformatics*, 12, 2018. ISSN 1662-5196. doi: 10.3389/fninf.2018.00068.

[23] Dimitri Plotnikov, Inga Blundell, Tammo Ippen, Jochen Martin Eppler, Bernhard Rumpe, and Abigail Morrison. NESTML: a modeling language for spiking neurons. In *Lecture Notes in Informatics (LNI)*, volume P-254, pages 93–108. Modellierung 2016, Karlsruhe (Germany), 17 Mar 2016 - 19 Mar 2016, Gesellschaft für Informatik e.V. (GI), Mar 2016. URL `https://juser.fz-juelich.de/record/826510`.

[24] NVIDIA Corporation. *CUDA C++ Programming Guide*, 2019. URL `https://docs.nvidia.com/cuda/pdf/CUDA{\_}C{\_}Programming{\_}Guide.pdf`.

[25] Guibin Wang, Yi Song Lin, and Wei Yi. Kernel fusion: An effective method for better power efficiency on multithreaded GPU. *Proceedings - 2010 IEEE/ACM International Conference on Green Computing and Communications, GreenCom 2010, 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom 2010*, pages 344–350, 2010. doi: 10.1109/GreenCom-CPSCom.2010.102.

[26] Marcel Stimberg, Romain Brette, and Dan F.M. Goodman. Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:1–41, 2019. ISSN 2050084X. doi: 10.7554/eLife.47314.

[27] Shigeru Shinomoto, Hideaki Kim, Takeaki Shimokawa, Nanae Matsuno, Shintaro Funahashi, Keisetsu Shima, Ichiro Fujita, Hiroshi Tamura, Taijiro Doi, Kenji Kawano, Naoko Inaba, Kikuro Fukushima, Sergei Kurkin, Kiyoshi Kurata, Masato Taira, Ken Ichiro Tsutsui, Hidehiko Komatsu, Tadashi Ogawa, Kowa Koida, Jun Tanji, and Keisuke Toyama. Relating neuronal firing patterns to functional differentiation of cerebral cortex. *PLoS Computational Biology*, 5(7), 2009. ISSN 1553734X. doi: 10.1371/journal.pcbi.1000433.

[28] Eugene M Izhikevich and Gerald M Edelman. Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 105(9):3593–8, 2008. ISSN 1091-6490. doi: 10.1073/pnas.0712231105. URL `http://www.pnas.org/content/105/9/3593.abstract`.

[29] Tobias C. Potjans and Markus Diesmann. The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. *Cerebral Cortex*, 24(3):785–806, mar 2014. ISSN 1460-2199. doi: 10.1093/cercor/bhs358. URL `http://www.ncbi.nlm.nih.gov/pubmed/23203991https://academic.oup.com/cercor/article-lookup/doi/10.1093/cercor/bhs358`.

[30] Joana Cabral, Morten L. Kringelbach, and Gustavo Deco. Exploring the network dynamics underlying brain activity during rest. *Progress in Neurobiology*, 114:102–131, 2014. ISSN 18735118. doi: 10.1016/j.pneurobio.2013.12.005. URL `http://dx.doi.org/10.1016/j.pneurobio.2013.12.005`.

[31] Andrei Belitski, Arthur Gretton, Cesare Magri, Yusuke Murayama, Marcelo A. Montemurro, Nikos K. Logothetis, and Stefano Panzeri. Low-frequency local field potentials and spikes in primary visual cortex convey independent visual information. *Journal of Neuroscience*, 28(22):5696–5709, 2008. ISSN 02706474. doi: 10.1523/JNEUROSCI.0009-08.2008.

[32] Maximilian Schmidt, Rembrandt Bakker, Claus C. Hilgetag, Markus Diesmann, and Sacha J. van Albada. Multi-scale account of the network structure of macaque visual cortex. *Brain Structure and Function*, 223(3):1409–1435, 2018. ISSN 18632661. doi: 10.1007/s00429-017-1554-4.

[33] Rembrandt Bakker, Thomas Wachtler, and Markus Diesmann. CoCoMac 2.0 and the future of tract-tracing databases. *Frontiers in Neuroinformatics*, 6:1–6, 2012. ISSN 16625196. doi: 10.3389/fninf.2012.00030.

[34] N. T. Markov, M. M. Ercsey-Ravasz, A. R. Ribeiro Gomes, C. Lamy, L. Magrou, J. Vezoli, P. Misery, A. Falchier, R. Quilodran, M. A. Gariel, J. Sallet, R. Gamanut, C. Huissoud, S. Clavagnier, P. Giroud, D. Sappey-Marinier, P. Barone, C. Dehay, Z. Toroczkai, K. Knoblauch, D. C. Van Essen, and H. Kennedy. A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cerebral Cortex*, 24(1):17–36, 2014. ISSN 10473211. doi: 10.1093/cercor/bhs270.

[35] Mária Ercsey-Ravasz, Nikola T. Markov, Camille Lamy, David C. Van Essen, Kenneth Knoblauch, Zoltán Toroczkai, and Henry Kennedy. A Predictive Network Model of Cerebral Cortical Connectivity Based on a Distance Rule. *Neuron*, 80(1):184–197, 2013. ISSN 08966273. doi: 10.1016/j.neuron.2013.07.036.

[36] Nikola T. Markov, Julien Vezoli, Pascal Chameau, Arnaud Falchier, René Quilodran, Cyril Huissoud, Camille Lamy, Pierre Misery, Pascale Giroud, Shimon Ullman, Pascal Barone, Colette Dehay, Kenneth Knoblauch, and Henry Kennedy. Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex. *Journal of Comparative Neurology*, 522(1):225–259, 2014. ISSN 00219967. doi: 10.1002/cne.23458.

[37] Tom Binzegger, Rodney J. Douglas, and Kevan A.C. Martin. A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience*, 24(39):8441–8453, 2004. ISSN 02706474. doi: 10.1523/JNEUROSCI.1400-04.2004.

[38] Sacha Jennifer van Albada, Jari Pronold, A van Meegen, and Markus Diesmann. Usage and Scaling of an Open-Source Spiking Multi-Area Model of Monkey Cortex. In K Amunts, L Grandinetti, Thomas Lippert, and Nicolai Petkov, editors, *Brain-Inspired Computing*. Springer, in press.

[39] David Freedman and Persi Diaconis. On the histogram as a density estimator: L_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 57(4):453–476, 1981.

[40] Joseph M Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation*, 19(11): 2881–912, nov 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.11.2881. URL http://www.ncbi.nlm.nih.gov/pubmed/17883345.

[41] Claudia Clopath, Lars Büsing, Eleni Vasilaki, and Wulfram Gerstner. Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nature neuroscience*, 13:344–352, 2010. ISSN 1097-6256. doi: 10.1038/nn.2479. URL http://dx.doi.org/10.1038/nn.2479.

[42] Tim P Vogels and L. F Abbott. Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons. *The Journal of Neuroscience*, 25(46):10786–10795, 2005. ISSN 0270-6474, 1529-2401. doi: 10.1523/JNEUROSCI.3508-05.2005. URL http://www.jneurosci.org/content/25/46/10786.