Larger GPU-accelerated brain simulations with procedural connectivity

James C Knight[a,1] **and Thomas Nowotny**[a]

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

This manuscript was compiled on April 24, 2020

Large-scale simulations of spiking neural network models are an important tool for improving our understanding of the dynamics and ultimately the function of brains. However, even small mammals such as mice have on the order of 1×10^{12} synaptic connections which, in simulations, are each typically charaterized by at least one floating-point value. This amounts to several terabytes of data – an unrealistic memory requirement for a single desktop machine. Large models are therefore typically simulated on distributed supercomputers which is costly and limits large-scale modelling to a few privileged research groups. In this work, we describe extensions to GeNN – our Graphical Processing Unit (GPU) accelerated spiking neural network simulator – that enable it to 'procedurally' generate connectivity and synaptic weights 'on the go' as spikes are triggered, instead of storing and retrieving them from memory. We find that GPUs are well-suited to this approach because of their raw computational power which, due to memory bandwidth limitations, is often under-utilised when simulating spiking neural networks. We demonstrate the value of our approach with a recent model of the Macaque visual cortex consisting of 4.13×10^6 neurons and 24.2×10^9 synapses. Using our new method, it can be simulated on a single GPU – a significant step forward in making large-scale brain modelling accessible to many more researchers. Our results match those obtained on a supercomputer and the simulation runs $35\,\%$ faster on a single high-end GPU than previously on over 1000 supercomputer nodes.

spiking neural networks | GPU | high-performance computing | brain simulation

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 'weight' of synapses from each spiking presynaptic neuron to the input currents of connected postsynaptic neurons. Typically, the information describing which neurons are connected by a synapse and with what weight is generated before a simulation is run and stored in large arrays in random access memory (RAM). This creates high memory requirements for large-scale brain models, 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 constant, even when a simulation is distributed across thousands of nodes (4). However, high performance computer systems are bulky, expensive and consume a lot of power and are hence typically shared resources only accessible to a limited number of researchers for time-limited investigations.

Neuromorphic systems (5–9) 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 – throughout the entire system. In neuromorphic systems, this often translates to dedicating a large proportion of each chip to memory. 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 (9), 4 (5) or even 1 bit (7). 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). Unlike most other neuromorphic systems, the SpiNNaker (6) neuromorphic supercomputer is fully programmable and combines large on-chip memory with 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 limited and a physically large system is still needed even for moderately-sized simulations (9 boards for a model with around 10×10^3 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 (ALUs). 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 com-

Significance Statement

Simulations are an important tool for investigating how brains work. However, in order to faithfully reproduce some of the features found in biological brains, large models are required. Simulating such models has, until now, required so much memory that it could only be done on large, expensive supercomputers. In this work, we present a new method for simulating large models that significantly reduces memory requirements. This method is particularly well-suited for use on Graphical Processing Units (GPUs), which are a common fixture in many workstations. We demonstrate that using our new method we can not only simulate a very large brain model on a single GPU, but also do so $35\,\%$ faster than in previous supercomputer simulations.

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.

The authors declare no conflict of interest.

[1]To whom correspondence should be addressed. E-mail: J.C.Knight@sussex.ac.uk

pletely 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 is likely to require accessing around 8 B of memory and performing many fewer than the required 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 10×10^3 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 truly large-scale brain models and, although small numbers of GPUs can be connected using the high-speed NVLink (13) interconnect, beyond such small GPU clusters, scaling will be dictated by the same communication overheads as for other CPU-based distributed systems.

In this work, we present a novel approach that converts large-scale brain simulation from a problem which is memory-bound on a GPU to one where the large amount of available computational power can be used to reduce both memory and memory bandwidth requirements and enable large-scale brain simulations on a single GPU workstation.

Results

In the following subsections, we first present two recent innovations in our GeNN simulator (14) which enable simulations of very large models on a GPU. We then demonstrate the power of the new features by simulating a recent model of the Macaque visual cortex (15) with 4.13×10^6 neurons and 24.2×10^9 synapses.

Procedural connectivity. The first crucial innovation that enables large-scale simulations on a GPU is what we call '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 (16) which simulate their behaviour over time. Our GeNN simulator (14) 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 and GPU memories 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 by fetching its 8 B of parameter values 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 and is applicable whenever synapses are static – plastic synapses which change their weights during a simulation will have to be simulated in the traditional way. Although a similar approach was used by Eugene Izhikevich for simulating an extremely large thalamocortical model with 1×10^{11} neurons and 1×10^{15} synapses on a modest PC cluster in 2005 (17) – an incredible achievement – it has not been subsequently applied to modern hardware.

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 for all outgoing synapses of each neuron that fires a spike and immediately use the identified 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' which is described by a fixed probability P_{conn} that a neuron in the presynaptic population will be connected to a neuron in the postsynaptic population. In this case, the postsynaptic targets of any presynaptic neuron can 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 $\text{Unif}[0, 1]$ and generate a synapse if the sample is less than P_{conn}. However, for sparse connectivity, it is much more efficient to sample from the geometric distribution $\text{Geom}[P_{\text{conn}}]$ which is the distribution of the number of Bernoulli trials required to get the next success (i.e. a synapse). The geometric distribution can be sampled in constant time by inverting the cumulative density function (CDF) of the equivalent continuous distribution (the exponential distribution) to obtain $\frac{log(\text{Unif}[0,1])}{log(1-P_{\text{conn}})}$ (18, p499). Note, that when directly drawing from the uniform distribution, the sampling for each potential synapse is independent from any other potential synapse and all these operations could be performed in parallel. 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 number total connector'. In this scheme the synaptic connections between two neuronal populations are characterised by a fixed total number N_{syn} of randomly placed synapses. 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 $\text{Mult}[N_{\text{syn}}, \{P_n, P_n, \ldots, P_n\}]$, where $P_n = \frac{1}{N_{\text{pre}}}$, on the host CPU up front 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 $\text{Unif}[0, N_{\text{post}}]$ 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

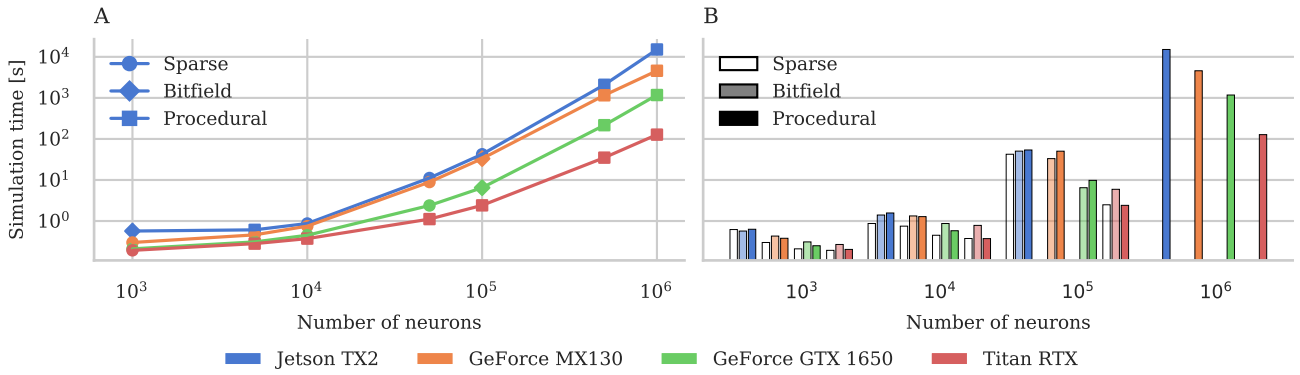

Fig. 1. Simulation time performance scaling on a range of modern GPUs (colors). **A** The best performing approach at each scale on each GPU (indicated by the symbols). For the largest models, the procedural method is always best. **B** Raw performance of each approach on each GPU. Missing bars indicate insufficient memory to simulate.

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 'convential' 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 (19). 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 random numbers. In constrast 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 used a network that was initially designed to investigate signal propagation through cortical networks (20), but subsequently has been widely used as a scalable benchmark (16). The network consists of N integrate-and-fire neurons, partitioned into $\frac{4N}{5}$ excitatory and $\frac{N}{5}$ inhibitory neurons. The two populations of neurons are connected to each other and with themselves with a fixed $P_{\text{conn}} = 10\,\%$ connection probability.

We ran simulations of this network at scales ranging from 1×10^3 to 1×10^6 neurons (100×10^3 and 100×10^9 synapses respectively) on a representative selection of modern NVIDIA 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. In Fig. 1 we compare the duration of these simulations using our new procedural approach against the standard approach of storing synaptic connections in memory using 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. None of the 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, the performance of the precedural approach is competitive with and sometimes better than the standard approach even at smaller scales. 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' representation cannot be used and the memory constraints for 'sparse' representations become even more severe.

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 approach presented in the previous section allows us to simulate models which would otherwise not fit into the memory of a single GPU, there are additional problems when using code generation for models with a large number of neuron and synapse populations. GeNN and all other SNN simulators which use code generation to generate all of their simulation code (21) (as opposed to, for example NESTML (22), which uses code generation only to generate neuron simulation code) generate seperate 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 (23, p278)). Therefore, in GeNN, multiple neuron populations are simulated within each kernel, resulting in code of the form shown in the following pseudocode which illustrates how 3 populations of 100 neurons each can

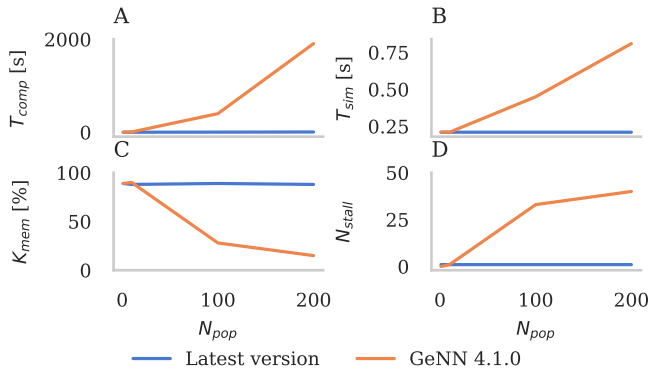

Fig. 2. Performance of a simulation of 1 000 000 LIF neurons driven by a gaussian input current, partitioned into varying numbers (N_{pop}) of populations and running on a workstation equipped with a Titan RTX GPU. **A** Compilation time (T_{comp}) using GCC 7.5.0. **B** Simulation time (T_{sim}) for an 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}).

be simulated in a single kernel:

```
void updateNeurons() {
  if(thread < 100) {
    // Update neuron population A
  } else if(thread >= 100 && thread < 200) {
    // Update neuron population B
  } else if(thread >= 200 && thread < 300) {
    // Update neuron population C
  }
}
```

This works well for a small number of populations but, as Fig. 2A illustrates, when we partition a model consisting of 1 000 000 LIF neurons into an increasingly large number of (smaller and smaller) populations, compilation time increases super-linearly and quickly becomes impractical. Furthermore, as Fig. 2B shows, the simulation also runs much more slowly when the model is partitioned into a large number of populations. Normally, we would expect this model to be memory bound as each thread in the model reads 32 B of data 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 (24) – 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 a larger number 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. The profiler documentation suggests that these particular events are likely to be caused by "Excessively jumping across large blocks of assembly code" (24, p47), which makes sense when we are generating kernels with hundreds of thousands of lines of code. Several neural modelling tools including Brian2 (25) 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 first '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, 100, 200};
```

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.

The multi-area model. Due to lack of computing power and sufficiently detailed connectivity data, previous models of the cortex have either focussed on modelling individual local microcircuits at the level of individual cells (27, 28) or modelling multiple connected areas at a higher level of abstraction (29). However, recent data (30) 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 (15, 31) is an example of such multi-scale modeling. It uses scaled versions of a previous, 4 layer microcircuit model (28) 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 (32) database, further refined using additional anatomical data (33) and heuristics (34) 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 (35) and cell-type-specific dendritic densities (36). Individual populations are connected by the fixed number connectors described above. For a full

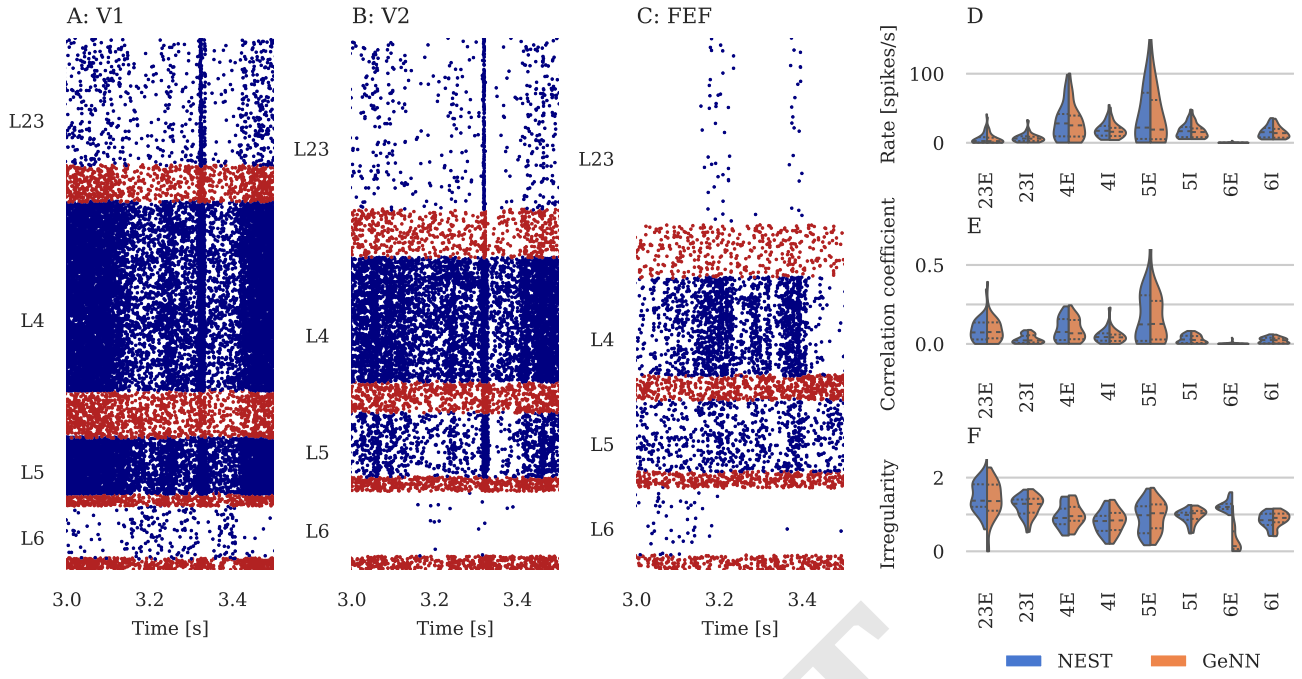

Fig. 3. Results of full-scale multi-area model simulation of resting state. **A-C** Raster plots of spiking activity of 3 % of the neurons in area V1 **A**, V2 **B**, and FEF **C**. Blue: excitatory neurons, red: inhibitory neurons. **D-F** Spiking statistics for each population across all 32 areas simulated using GeNN and NEST shown as split violin plots. Solid lines: medians, Dashed lines: Interquartile range (IQR). **D** Population-averaged firing rates. **E** Average pairwise correlation coefficients of spiking activity. **F** Irregularity measured by revised local variation LvR (26) averaged across neurons.

description of the multi-area model please see (15, 31). In 2018, this model was simulated using NEST (2) on one rack of an IBM Blue Gene/Q supercomputer (a 2 m high enclosure containing 1024 compute nodes, weighing over 2 t and requiring around 80 kW of power). On this system, initialization of the model took around 5 min and simulating 1 s of biological time took approximately 12 min (15).

The multi-area model consists of 4.13×10^6 neurons in 254 populations and 24.2×10^9 synapses in 64 516 populations. 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 independant 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 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 validate our GeNN simulations, we ran a 10.5 s simulation of the multi-area 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 7.7 min in the ground state and 8.4 min in the resting state– 35 % and 30 % less than the supercomputer simulation respectively. Fig. 3A-C shows some example spike rasters from the resting state simulation of three of the modelled areas, illustrating the characteristic irregular activity and population bursts. 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 (26), 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 violin plots in Fig. 3D-F show the comparison of the distributions of values obtained from the two simulations – which are essentially identical.

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 (15, 31) on a single modern GPU. While this represents a significant step forward in terms of making truly large-scale brain modelling tools accesible 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 (described in our previous work (12)). These are a per-neuron (rather

Table 1. Model parameters.

Parameter	Procedural connectivity benchmark	Merging benchmark	Multi-area model
τ_{m} [ms]	20	20	2
V_{rest} [mV]	-60.0	-70.0	-65
V_{th} [mV]	-50.0	-51.0	-50
R_{m} [MΩ]	20	20	40
τ_{syn} [ms]	5/10[1]	—	0.5
τ_{ref} [ms]	5	2	2
I_{ext_j} [nA]	0.55	1.00 ± 0.25	Poisson[2]
w_{ij} [nA]	$\frac{3.2}{N}$ / $\frac{40.8}{N}$ [1]	—	Various[2]

[1] Excitatory/Inhibitory.
[2] Please refer to original works (15, Table 1,2)

than per-synapse) data structure but, because the inter-area connections in the model have delays of up to 500 simulation timesteps (0.1 ms), the delay buffers become quite large.

One important aspect of large-scale brain simulations not addressed in this work is synaptic plasticity and its role in learning. As discussed in our previous work (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 (37, 38) 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 our previous work (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. Key components such as the random number generator could be implemented directly in hardware leading to truly game-changing compute time improvements.

Materials and Methods

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

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

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_j} represents the synaptic input current and I_{ext_j} 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, pre-synaptic spikes lead to exponentially-decaying input currents I_{syn_j}

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

where τ_{syn} represents the decay time constant and t_j are the arrival times of incoming spikes from n presynaptic neurons. The continuous terms of the Eq. 1 and 2 are seperately solved algebraically so that the synaptic input current I_{in_i} is treated as a constant throughout each simulation timestep.

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).

1. Herculano-Houzel S, Mota B, Lent R (2006) Cellular scaling rules for rodent brains. *Proceedings of the National Academy of Sciences* 103(32):12138–12143.
2. Gewaltig MO, Diesmann M (2007) NEST (NEural Simulation Tool). *Scholarpedia* 2(4):1430.
3. Carnevale NT, Hines ML (2006) *The NEURON book.* (Cambridge University Press).
4. Jordan J, et al. (2018) Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. *Frontiers in Neuroinformatics* 12(February):2.
5. Frenkel C, Lefebvre M, Legat JD, Bol D (2018) 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* PP(XX):1–1.
6. Furber SB, Galluppi F, Temple S, Plana LA (2014) The SpiNNaker Project. *Proceedings of the IEEE* 102(5):652–665.
7. Merolla PA, et al. (2014) A million spiking-neuron integrated circuit with a scalable communication network and interface. *{S}cience* 345(6197):668–673.
8. Qiao N, et al. (2015) A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience* 9(APR):1–17.
9. Schemmel J, Kriener L, Muller P, Meier K (2017) An accelerated analog neuromorphic hardware system emulating NMDA- and calcium-based non-linear dendrites. *Proceedings of the International Joint Conference on Neural Networks* 2017-May:2217–2226.
10. van Albada SJ, Helias M, Diesmann M (2015) Scalability of Asynchronous Networks Is Limited by One-to-One Mapping between Effective Connectivity and Correlations. *PLoS Computational Biology* 11(9):1–37.
11. Rhodes O, et al. (2020) Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378(2164):20190160.
12. Knight JC, Nowotny T (2018) GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model. *Frontiers in Neuroscience* 12(December):1–19.
13. NVIDIA Corporation (2020) NVLink Fabric Multi-GPU Processing.
14. Yavuz E, Turner J, Nowotny T (2016) GeNN: a code generation framework for accelerated brain simulations. *Scientific reports* 6(November 2015):18854.
15. Schmidt M, et al. (2018) A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. *PLoS Computational Biology* 14(10):1–38.
16. Brette R, et al. (2007) Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience* 23(3):349–98.
17. Izhikevich EM (2005) Large-Scale Simulation of the Human Brain.
18. Devroye L (2013) *Non-uniform random variate generation.* (Springer-Verlag New York, New York).
19. Salmon JK, Moraes MA, Dror RO, Shaw DE (2011) 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.* (ACM Press, New York, New York, USA), Vol. 81, p. 1.
20. Vogels TP, Abbott LF (2005) Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons. *The Journal of Neuroscience* 25(46):10786–10795.
21. Blundell I, et al. (2018) Code Generation in Computational Neuroscience: A Review of Tools and Techniques. *Frontiers in Neuroinformatics* 12(November).
22. Plotnikov D, et al. (2016) NESTML: a modeling language for spiking neurons. pp. 93–108.
23. NVIDIA Corporation (2019) CUDA C++ Programming Guide.
24. NVIDIA Corporation (2020) Nsight Compute.
25. Stimberg M, Goodman DFM, Benichoux V, Brette R (2014) Equation-oriented specification of neural models for simulations. *Frontiers in Neuroinformatics* 8(February):1–14.
26. Shinomoto S, et al. (2009) Relating neuronal firing patterns to functional differentiation of cerebral cortex. *PLoS Computational Biology* 5(7).
27. Izhikevich EM, Edelman GM (2008) Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences of the United States of America* 105(9):3593–8.
28. Potjans TC, Diesmann M (2014) The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. *Cerebral Cortex* 24(3):785–806.
29. Cabral J, Kringelbach ML, Deco G (2014) Exploring the network dynamics underlying brain activity during rest. *Progress in Neurobiology* 114:102–131.
30. Belitski A, et al. (2008) Low-frequency local field potentials and spikes in primary visual cortex convey independent visual information. *Journal of Neuroscience* 28(22):5696–5709.
31. Schmidt M, Bakker R, Hilgetag CC, Diesmann M, van Albada SJ (2018) Multi-scale account of the network structure of macaque visual cortex. *Brain Structure and Function* 223(3):1409–1435.
32. Bakker R, Wachtler T, Diesmann M (2012) CoCoMac 2.0 and the future of tract-tracing databases. *Frontiers in Neuroinformatics* 6(DEC):1–6.
33. Markov NT, et al. (2014) A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cerebral Cortex* 24(1):17–36.

34. Ercsey-Ravasz M, et al. (2013) A Predictive Network Model of Cerebral Cortical Connectivity Based on a Distance Rule. *Neuron* 80(1):184–197.
35. Markov NT, et al. (2014) Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex. *Journal of Comparative Neurology* 522(1):225–259.
36. Binzegger T, Douglas RJ, Martin KA (2004) A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience* 24(39):8441–8453.
37. Brader JM, Senn W, Fusi S (2007) Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation* 19(11):2881–912.
38. Clopath C, Büsing L, Vasilaki E, Gerstner W (2010) Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nature neuroscience* 13(December 2009):344–352.