Large-scale brain simulations on the desktop using 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 March 31, 2020

Large-scale simulations of spiking neural networks are important for improving our understanding of the dynamics and ultimately function of brains. However, even small mammals such as mice have approximately 1×10^{12} synaptic connections which are typically charaterized by at least one floating point value per synapse. This amounts to several terabytes of connection data – an unrealistic memory requirement for a single desktop machine. Simulations of large spiking neural networks are therefore typically executed on large distributed supercomputers. This is costly and limits large-scale modelling to a select few research groups with the appropriate resources. In this work, we describe extensions to GeNN – our GPU-based 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, this model can be simulated on a single GPU. Our results match those obtained on a supercomputer and the simulation runs 35 % faster on a single high-end GPU than a previous simulation executed 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} (1) synapses which connect them. In computer simulations of spiking neural networks, propagating spikes through synapses involves reading a 'row' of synapses connecting a spiking presynaptic neuron to its postsynaptic partners and adding the 'weight' of each synapse in the row to a 'bin' containing the postsynatic neuron's input for the next simulation timestep. Typically, the information describing which neurons are connected by a synapse and with what conductance, is generated before a simulation is run and stored in large matrices 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 large amounts of power, meaning that they are typically shared resources that are only accessible to a limited number of researchers and for strongly time-limited investigations.

Neuromorphic systems (5–10) take inspiration from the brain and have been developed specifically for simulating large spiking neural networks. 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 meaning 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 (10), 4 (5) or even 1 bit (6, 8). Such strategies allow some classes of spiking neural networks to be simulated very efficiently, but reducing the degree of connectivity in large-scale brain simulations to fit within the constraints of current neuromorphic systems inevitably changes their dynamics (11). Unlike the majority of other neuromorphic systems, the SpiNNaker (7) neuromorphic super-computer is entirely programmable and combines a large amount of on-chip memory with external memories, distributed across the system for the storage of synaptic connectivity. SpiNNaker's external memory bandwidth, on-chip memory capacity and the computational power of each core are all tailored to large-scale brain simulation meaning that the output bins of the synapse processing algorithm can fit in on-chip memory and there is enough external memory bandwidth to fetch synaptic rows fast enough for real-time simulation of large-scale models (12). This is a promising approach for future research but, because of its prototype nature, the availability of SpiNNaker hardware is

Significance Statement

Brain simulations are an important tool for investigating how the brain works. 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 Unit (GPU) accelerators, which are a common fixture in many workstations. We demonstrate that using our new method we can simulate a very large brain model on a single GPU, and 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.Knightsussex.ac.uk

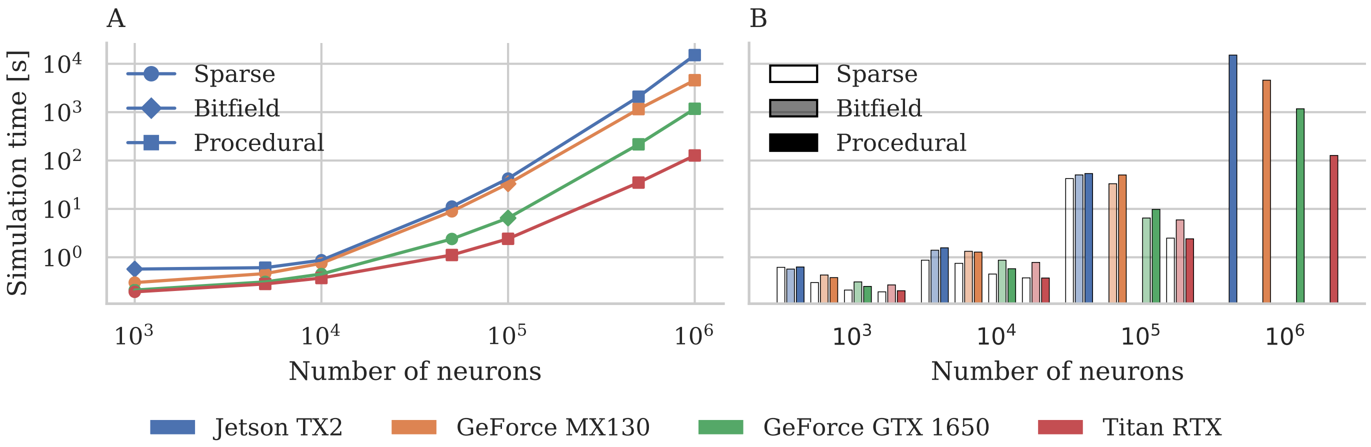

Fig. 1. Simulation time performance scaling on a range of modern GPUs (colors). **A** The best performing approach at each scale on each GPU (indicated by the symbols). For the largest models, the procedural method is always best. **B** Raw performance of each approach on each GPU. Missing bars indicate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

limited and a physically large system is still required for even moderately-sized simulations (9 boards for a simulation with around 10×10^3 neurons and 300×10^6 synapses (12)).

Modern GPUs have relatively small amounts of on-chip memory and, instead, dedicate the majority of their silicon area to arithmetic logic units (ALUs). GPUs use dedictated 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, processing a synapse in a spiking neural network simulation is likely to require accessing approximately 8 B of memory and performing many fewer than the required 80 instructions. This makes synaptic updates highly memory bound. Nonetheless, we have shown in previous work (13) that, as GPUs have significantly higher total memory bandwidth than even the most expensive 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 requirements. However, individual GPUs do not have enough memory to simulate truly large-scale brain models and, although small numbers of GPUs can be connected together using the high-speed NVLink (14) interconnect, beyond such small GPU clusters, scaling will be dictated by the same communication overheads as for other MPI-based distributed systems.

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

Results

In the following subsections, we first present two recent innovations in our GeNN simulator (15) which allow us to use it for simulating very large models on a single GPU. We then demonstrate the power of these new features by simulating a recent model of the Macaque visual cortex (16) consisting of 4.13×10^6 neurons and 24.2×10^9 synapses on a single GPU. We find that we not only obtain the same results as in a previous simulation on a high-performance supercomputer, but that our simulation also runs faster.

Procedural connectivity. The first crucial innovation to enable 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, eventually, are translated into time or event-driven update algorithms (17) that calculate their state from previous states, allowing us to simulate their behaviour over time. Our GeNN simulator (15) 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 – often involving random number generation – once prior to the simulation on the main CPU. The results are stored in CPU and GPU memories and used throughout the simulation. We have recently extended GeNN to also use code generation to generate efficient, parallel model initialisation methods that run on the GPU from initialisation code snippets (13). Offloading initialisation to the GPU in this way sped up model initialisation by around 20× on a desktop PC (13), demonstrating that initialisation algorithms are well-suited to 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 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. Although 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 **(TODO: cite)** – an incredible achievement – it has not been applied to more modern hardware since.

We implemented procedural connectivity as an option in GeNN by repurposing our previously developed parallel initialisation methods. Instead of being run once for all synapses at the beginning of the simulation, when using procedural connectivity, the methods are rerun during the simulation for the outgoing synapses of each neuron that fires a spike. The identified connections and weights are then used to run the post-synaptic code that calculates the effect of the spike onto other neurons. This is possible because 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 $\mathrm{Unif}[0,1]$ and generate a synapse if the sample is less than P_{conn}. However, it is much more efficient to sample from the geometric distribution $\mathrm{Geom}[P_{\mathrm{conn}}]$ 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 $\frac{log(\mathrm{Unif}[0,1])}{log(1-P_{\mathrm{conn}})}$ (18, p499). Note, that when directly drawing from the uniform distribution, the sampling for each potential synapse is completely independent from any other potential synapse and all these operations could be performed in parallel. However, for the more efficient 'beta-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 connectivity scheme for defining connectivity is the fixed number 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 subset of the N_{syn} synapses which connect each presynaptic neuron must be calculated by sampling from the multinomial distribution $\mathrm{Mult}[N_{\mathrm{syn}}, \{P_{row}, P_{row}, \ldots, P_{row}\}]$ where $P_{row} = \frac{N_{\mathrm{post}}}{N_{\mathrm{pre}}N_{\mathrm{post}}} = \frac{1}{N_{\mathrm{pre}}}$, because the numbers for different presynaptic neurons are not independent from each other. This operation cannot be efficiently parallelised so we perform it on the host CPU and store the results in a GPU memory array. 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 $\mathrm{Unif}[0, N_{\mathrm{post}}]$. Note, that this can only be done because the targets of each presynaptic neuron are independent from those of any other pre-synaptic 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 'convential' random number generators (RNGs), but each presynaptic neuron would need to maintain its own RNG state which would lead to a significant memory overhead. Instead, we use a 'counter-based' Philox4×32-10 RNG (19). Counter-based RNGs are designed for parallel applications and essentially consist of a pseudo-random bijective function which takes a counter as an input (in this case a 128 bit number) and outputs random numbers. In constrast to convential RNGs, 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 (17). 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 $P_{\mathrm{conn}} = 10\,\%$ 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 designed for robotic applications with 8 GB of shared memory

Geforce MX130 a laptop GPU with 2 GB of dedicated memory

Geforce GTX 1650 a low-end desktop GPU with 4 GB of dedicated memory

Titan RTX a high-end workstation GPU with 24 GB of dedicated memory

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 (13) 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 $N_{\mathrm{pre}} \times N_{\mathrm{post}}$ array of bits where a '1' at position i, j indicates the existence of a synapse between neurons i and j and '0' its absence. None of the devices used 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 its performance is competitive with and sometime better than the standard approach. All of the excitatory and inhibitory synapses in this model have the same synaptic weight meaning that they can be hard-coded into the procedural connectivity kernels. However, if the weights vary across synapses, the 'bitfield' representation cannot be used

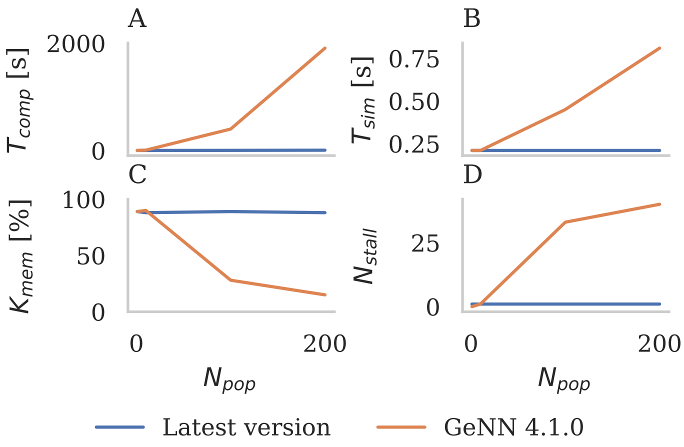

Fig. 2. Performance of a simulation of 1 000 000 LIF neurons driven by a gaussian input current, partitioned into varying numbers (N_{pop}) of populations and running on a workstation equipped with a Titan RTX GPU. **A** Compilation time (T_{comp}) using GCC 7.5.0. **B** Simulation time (T_{sim}) for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

and the memory constraints for the 'sparse' representation 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 code is organised into these kernels. While the procedural connectivity approach presented in the previous section allows us to simulate models which would otherwise not fit within the memory of a single GPU, there are additional problems when using code generation to generate simulation code for models with a large number of neuron and synapse populations.

GeNN and – to the best of our knowledge (21) – all other SNN simulators which use code generation to generate all of their simulation code (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 approach 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 1000 neurons each could be simulated in a single kernel:

```
void updateNeurons()
{
  if(thread < 1000) {
    // Update neuron population A
  }
  else if(thread >= 1000 && thread < 2000) {
    // Update neuron population B
  }
  else if(thread >= 2000 && thread < 3000) {
    // Update neuron population C
  }
}
```

This approach works well for models with 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 as the size of the neuron kernel increases – quickly becoming 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 we discussed previously, hiding the latency of these memory accesses would require approximately 320 arithmetic operations which is many more than are required to sample from the uniform 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 using the profiler, we found that this drop in performance was accompanied by an increasing number of "No instruction" stalls as shown in Fig. 2D. Stalls are events which prevent the GPU from doing any work during a clock cycle and 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.

To address these issues, 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] = {1000, VA};
neuronUpdateGroup[1] = {1000, VB};
neuronUpdateGroup[2] = {1000, 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

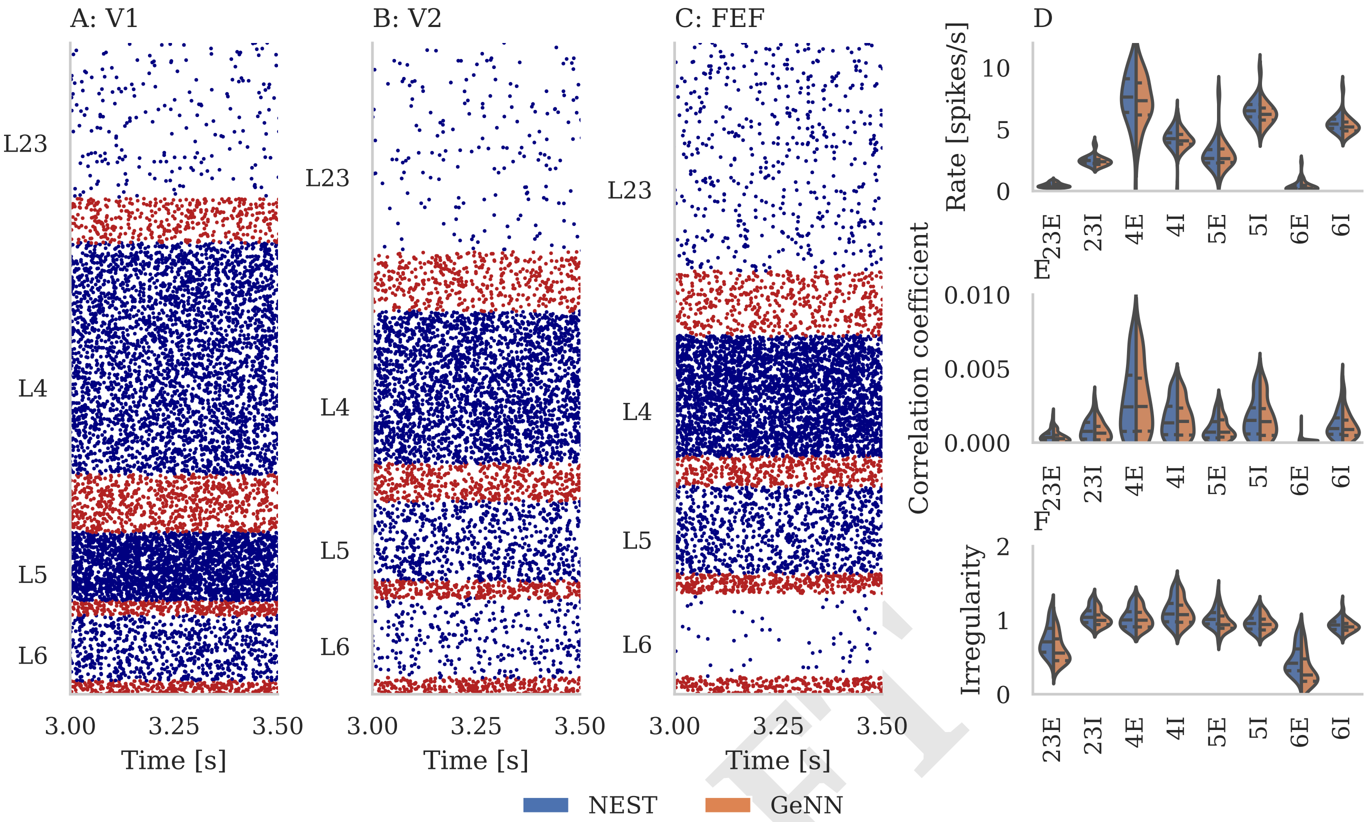

Fig. 3. Results of full-scale multi-area model simulation. **A-C** Raster plots of spiking activity of 3 % of the neurons in area V1 **A**, V2 **B**, and FEF **C**. Blue: excitatory neurons, red: inhibitory neurons. **D-F** Spiking statistics for each population across all 32 areas simulated using GeNN 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 ofspiking activity. **F** Irregularity measured by revised local variation LvR (25) averaged across neurons.

a cumulative sum of threads used for each population. Each thread performs a simple binary search within this array to find the index of the neuron and population it should simulate:

```
unsigned int startThread[3] = {0, 1000, 2000};
void updateNeurons()
{
   if(thread < 3000) {
      // Binary search in startThread to
      // determine which neuron in which
      // population should be processed.
      // Then update, accessing variables
      // through neuronUpdateGroup
   }
}
```

As Fig. 2 shows, this approach solves the issues with compilation time and simulation performance caused by large numbers of populations.

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 (26, 27) or modelling multiple connected areas at a higher level of abstraction where entire ensembles of neurons are described by a small number of differential equations (28). However, data from several species **(TODO: find citation)** 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 (16, 29) is an example of such multi-scale modeling – using scaled versions of a previous, 4 layer microcircuit model (27) to implement 1 mm^2 'patches' for each of 32 areas of the macaque visual cortex. The 32 areas are connected together with connectivity based on inter-area axon tracing data from the CoCoMac (30) database, further refined using additional anatomical data (31) and heuristics (32) to obtain estimates for the number of synapses between areas. These synapses are distributed between populations in the source and target area using layer-specific tracing data (33) and cell-type-specific dendritic densities (34). Individual populations are connected by the fixed number connectors described previously. For a full description of the construction of the multi-area model please refer to the original works (16, 29). 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 and weighing over 2 t). On this system, initialization of the model took around 5 min and simulating 1 s of biological time took approximately 12 min (16).

The multi-area model consists of 4.13×10^6 neurons split into 254 populations and 24.2×10^9 synapses split into 64 516 populations meaning that, without the kernel merging approach presented above the model would be unlikely to 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, meaning that the bitfield data structure cannot be used to represent the connectivity. Even if we assume that 16 bit floating point would provide sufficient weight precision, that delays could be expressed as an 8 bit integer and that the neuron populations are all small enough to be indexed using 16 bit indices, our sparse data structure would still require 5 B per synapse, meaning that this model's synaptic data would require 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, with the procedeural connectivity method, we are able to simulate this model on single workstation with one Titan RTX GPU with 24 GB of memory.

In order to validate our GeNN simulations of the multi-area model, we ran a 10.5 s simulation of the model. 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 – 35 % less than in the previous supercomputer simulation. 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. 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 previously published values of the same measures obtained from the supercomputer simulations. We calculated irregularity using the revised local variation LvR (25), 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 shows 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 (16, 29) 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 in total, with the majority (8.5 GB) being used for the implementation of the circular buffers used to simulate dendritic delay (described in more detail in our previous work (13)). 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 around 50 ms (500 0.1 ms timesteps), the delay buffers become extremely large.

One important aspect of large-scale brain simulations that we have not addressed in this work is synaptic plasticity and its role in learning. As discussed in our previous work (13), GeNN has support for 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 many synaptic plasticity rules that only require access to presynaptic spikes and postsynaptic neuron states such as membrane voltage (35, 36), but for many Spike-Timing-Dependent Plasticity (STDP) rules access to *postsynaptic* spikes is also required. GeNN supports such rules by automatically generating a suitable lookup table structure (see our previous work (13) for more details) and this process could be adapted to generate a lookup table from procedural connectivity although 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.

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_{in_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(16, Table 1,2)

In this work, we have discussed the idea of procedural connectivity purely in the context of GPU hardware but, we believe that there is also some potential for developing new types of neuromorphic hardware built from the ground up for procedural connectivity. Key components such as the counting random number generator could be implemented directly in hardware leading to truly game-changing compute time improvements.

Materials and Methods

In all three experiments presented in this work, neurons are modelled as leaky integrate-and-fire (LIF) units using the parameters listed in Table 1. The membrane voltage V_i of neuron i is modelled as

$$\tau_{\text{m}} \frac{dV_i}{dt} = (V_i - V_{\text{rest}}) + R_{\text{m}} I_{\text{in}_j}, \qquad [1]$$

where τ_{m} and R_{m} represent the time constant and resistance of the neuron's cell membrane, V_{rest} defines the resting potential and I_{in_j} represents the input current. When the membrane voltage crosses a threshold V_{th} a spike is emitted, the membrane voltage is reset to V_{rest} and membrane potential integration 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_{in_j}

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

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} entering into equation 1 is treated as a constant during each simulation timestep.

ACKNOWLEDGMENTS. We like to thank Jari Pronold, Sacha van Albada and Maximilian Schmidt for their valuable assistance with using the data and analysis tools published alongside the multi-area model – without these contributions, our comparison with these results would not have been possible. 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-mmˆ2 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. Frenkel C, Legat Jd, Bol D (2019) A 65-nm 738k-Synapse/mm 2 Quad-Core Binary-Weight Digital Neuromorphic Processor with Stochastic Spike-Driven Online Learning in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. (IEEE), pp. 1–5.
7. Furber SB, Galluppi F, Temple S, Plana LA (2014) The SpiNNaker Project. *Proceedings of the IEEE* 102(5):652–665.
8. Merolla PA, et al. (2014) A million spiking-neuron integrated circuit with a scalable communication network and interface. *{S}cience* 345(6197):668–673.
9. 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.
10. 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.
11. 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.
12. Rhodes O, et al. (2019) Real-Time Cortical Simulation on Neuromorphic Hardware.
13. 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.
14. NVIDIA Corporation (year?) NVLink Fabric Multi-GPU Processing.
15. Yavuz E, Turner J, Nowotny T (2016) GeNN: a code generation framework for accelerated brain simulations. *Scientific reports* 6(November 2015):18854.
16. 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.
17. 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.
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. Shinomoto S, et al. (2009) Relating neuronal firing patterns to functional differentiation of cerebral cortex. *PLoS Computational Biology* 5(7).
26. 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.
27. 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.
28. Cabral J, Kringelbach ML, Deco G (2014) Exploring the network dynamics underlying brain activity during rest. *Progress in Neurobiology* 114:102–131.
29. 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.
30. Bakker R, Wachtler T, Diesmann M (2012) CoCoMac 2.0 and the future of tract-tracing databases. *Frontiers in Neuroinformatics* 6(DEC):1–6.
31. (2014) A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cerebral Cortex* 24(1):17–36.
32. Ercsey-Ravasz M, et al. (2013) A Predictive Network Model of Cerebral Cortical Connectivity Based on a Distance Rule. *Neuron* 80(1):184–197.
33. Markov NT, et al. (2014) Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex. *Journal of Comparative Neurology* 522(1):225–259.
34. 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.
35. 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.
36. 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.