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 18, 2020

Large-scale simulations of spiking neural networks have become important tools in helping us improve our understanding of the dynamics of brain circuits, and ultimately brain function. However, even small mammals such as mice have on the order of 1×10^{12} synaptic connections (?) which are typically charaterized by at least one floating point value per synapse, usually related to their synaptic conductance. If single precision floating point variables were used to store individual conductance values for all synapses of a mouse brain, several terabytes of memory would be required. Because such memory requirements are beyond what is plausible for a single desktop machine, simulations of large-scale spiking neural networks are currently typically simulated on large distributed high performance computing systems. This is costly and limits this level of modelling to a select few research groups who have access to appropriate supercomputing resources. However, large parts of current brain models are normally described by means of simple algorithms which determine the connectivity and the strengths of synaptic connections. 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, rather than generating large connectivity matrices up front and retrieving them from memory when needed, which is how practiacly all simulations are run today. We find that high-end GPUs are well-suited to this approach as they provide a large amount of raw computational power which is often underutilised when simulating spiking neural networks due to the limited memory bandwidth available to each parallel computing element. To demonstrate the value of this approach, we have implemented a recent model of the Macaque visual cortex consisting of 4.13×10^6 neurons and 24.2×10^9 synapses. With our new method this model can be simulated on a single GPU. We demonstrate that the results match results obtained on a supercomputer and that the simulation runs faster on a singleh high-end GPU than a previous simulation which was 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} 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 which neurons are connected by a synapse and with what weight, is generated before a simulation is run and stored in large matrices in random access memories. 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 (?) or NEURON (?). By careful design, these simulators can maintain the memory requirements for each node constant, even when a simulation is distributed across thousands of nodes (?). 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 (? ? ? ? ? ?) take inspiration from the brain and have been developed specifically as an alternative 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 (?), 4 (?) or even 1 bit (? ?). 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 (?). Unlike the majority of other neuromorphic systems, the SpiNNaker (?) 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

Significance Statement

Authors must submit a 120-word maximum statement about the significance of their research paper written at a level understandable to an undergraduate educated scientist outside their field of speciality. The primary goal of the Significance Statement is to explain the relevance of the work in broad context to a broad readership. The Significance Statement appears in the paper itself and is required for all research papers.

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

enough external memory bandwidth to fetch synaptic rows fast enough for real-time simulation of large-scale models (?). **(TODO: good argument against SpiNNaker)** This is a promising approach for future research but because of its prototype nature its availability is 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 (?)).

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 (?) 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. Individual GPUs do, however, 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 **(TODO: cite)** interconnect, scaling will be dictated by the same communication overheads as for other MPI-based distributed systems beyond such small GPU clusters.

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 we will first two recent innovations in our GeNN simulator (?) which allow to use it for simulating large-scale models on a single GPU. We will then demonstrate the power the new features by simulating a recent model of the Macaque visual cortex (?) 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 the previous simulation on a high-performance supercomputer, but our simulation also runs faster.

Procedural connectivity. Our GeNN simulator (?) uses code generation to convert neuron and synapse models – described using 'snippets' of C-like code – into CUDA code for GPU simulation. We previously extended GeNN to allow the same approach to be applied to generating efficient, parallel model initialisation code from code snippets describing the algorithms to use for initialising individual state variables and synaptic connectivity (?). Parallelising initialisation in this manner sped up model initialisation by around 20× on a desktop PC, but also indicates just how well-suited these initialisation algorithms are to GPU. In fact, it seems somewhat illogical to run these algorithms once only to fill the limited memory of the GPU with data and subsequently read it back throughout the simulation

To demonstrate the performance and scalability of this new approach, we ran several simulations of a network, initially designed as a medium for experimentation into signal propagation through cortical networks (?), but subsequently widely used as a scalable benchmark (?). The network consists of 10 000 integrate-and-fire neurons, split between an excitatory population of 8000 cells and an inhibitory population of 2000 cells.

Kernel merging. 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 large numbers of neuron and synapse populations.

GeNN and – to the best of our knowledge (?) – all other SNN simulators which use code generation to generate all of their simulation code (rather than, for example NESTML (?), which uses code generation to generate neuron simulation code) generate seperate pieces of code to simulate each population of neurons and synapses. This approach allows optimizations such as the hard-coding of constant parameters to be easily performed and, although generating code for models with many populations will result in large code size, C++ CPU code can be easily divided between multiple modules and compiled in parallel, minimizing the effect 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 **(TODO: cite)**). Therefore, in GeNN, multiple neuron populations are simulated within each kernel resulting in code such as the following example which illustrates how 3 populations of 1000 neurons 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 works well for models with small numbers of populations but, as figure 2A illustrates, compilation time increases super-linearly with the number of populations (i.e. the size of the neuron kernel) – quickly becoming impractical. Additionally, and even more critically, figure 2B shows that simulations of the same model, artificially divided into more populations, run much slower. Each thread in this model reads 32 B of data and, as we discussed previously, hiding the latency of these memory accesses would require approximately 320 arithmetic operations. Sampling from the uniform distribution and updating a LIF neuron requires many fewer operations than this so we would expect this kernel to be memory bound.

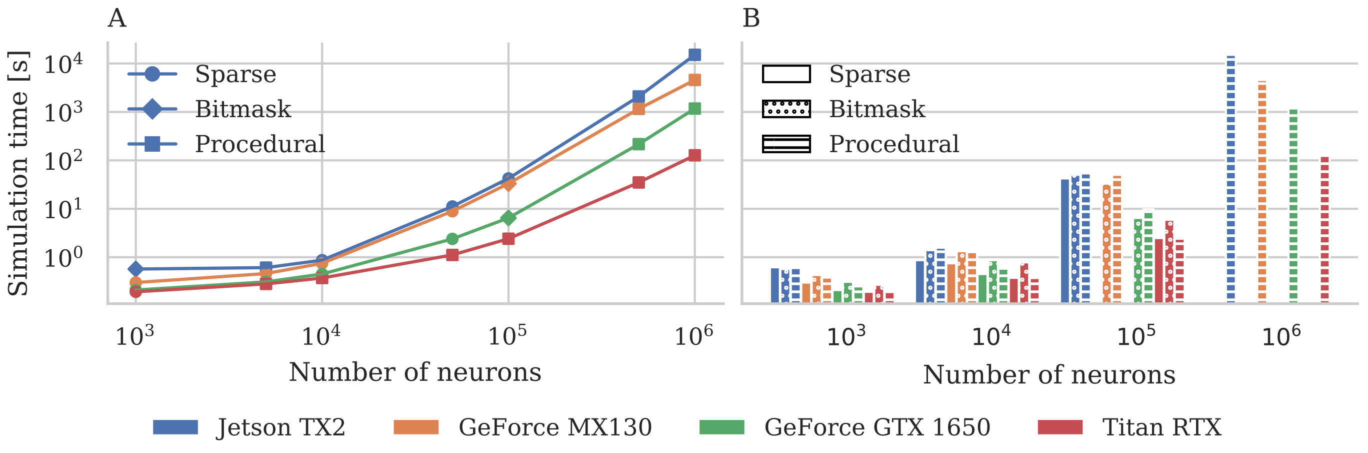

Fig. 1. Performance scaling on a range of modern GPUs. **A** The best performing approach at each scale. **B** Raw performance of each approach.

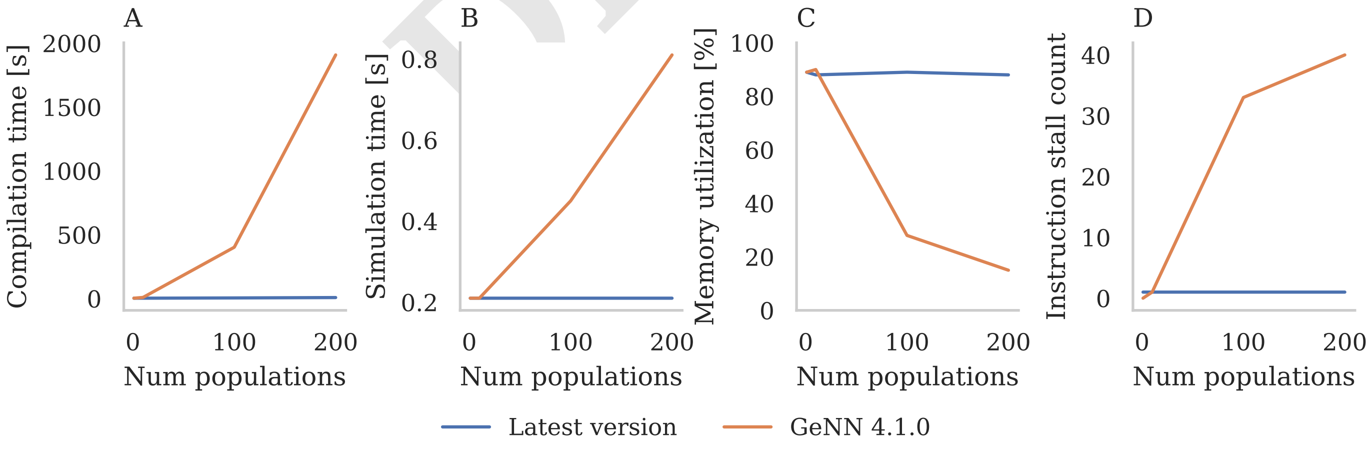

Fig. 2. Performance of a simulation of $1\,000\,000$ LIF neurons driven by a gaussian input current, partitioned into varying numbers of populations. **A** Compilation time using GCC 7.5.0. **B** Simulation time for an $1\,\mathrm{s}$ simulation. **C** Memory throughput reported by NVIDIA Nsight compute profiler 'Speed of light' metric. **D** Number of 'No instruction' stalls reported by NVIDIA Nsight compute profiler.

Figure 2C – obtained using data from the NVIDIA Nsight compute profiler **(TODO: cite)** – shows that this to be true with the memory system being around 90 % utilised for small numbers of populations. However, if the model is partitioned into large numbers of populations, the kernel stops being able to efficiently use the memory. Investigating the performance counters

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

NeuronUpdateGroup neuronUpdateGroup
{
    {1000, d_VA},
    {1000, d_VB},
    {1000, d_VC}
};

void updateNeurons()
{
    if(thread < 3000) {
        // Determine which population thread
        // should be processing and update using
        // variables in neuronUpdateGroup
    }
}
```

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 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 **(TODO: find citation)**. 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 be captured by modelling interconnected microcircuits at the level of individual cells.

By using a supercomputer to simulate a model based on the latest connectivity data and The multi-scale model of the macaque visual cortex (**?**) developed by

Discussion

- Further scaling - memory only required for neuron parameters
- Learning
- Hardware for procedural connectivity?

Materials and Methods

Please describe your materials and methods here. This can be more than one paragraph, and may contain subsections and equations as required. Authors should include a statement in the methods section describing how readers will be able to access the data in the paper.

- LIF neuron
- Exponential static synapses
- Connectivity
- Parameter values for scaling and merging experiments

Neuron models. Example text for subsection.

ACKNOWLEDGMENTS. Please include your acknowledgments here, set in a single paragraph. Please do not include any acknowledgments in the Supporting Information, or anywhere else in the manuscript.

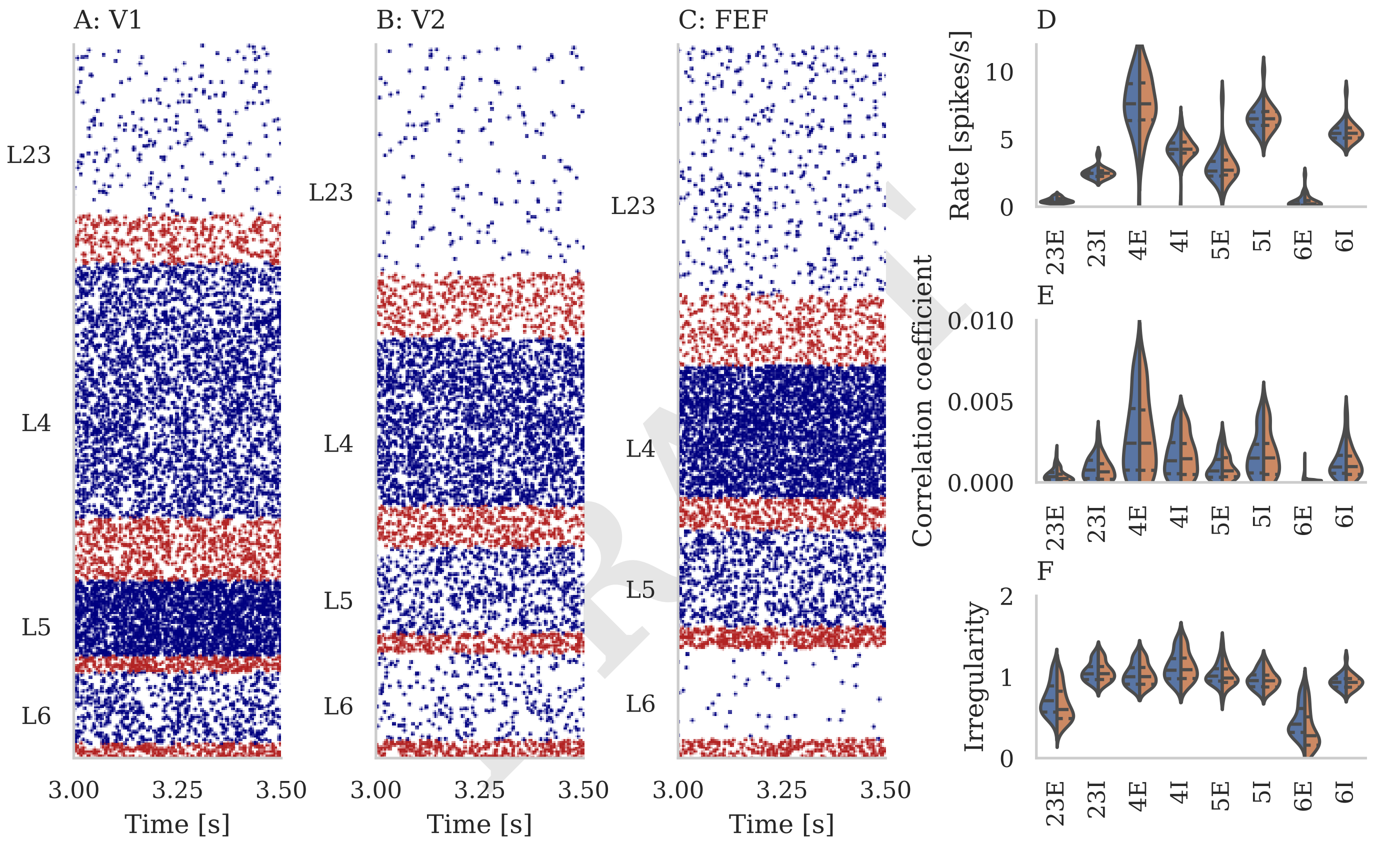

Fig. 3. Results of full-scale multi-area model simulation. **A-C** Raster plots of spiking activity of $3\,\%$ of the neurons in area V1 **A**, V2 **B**, and FEF **C**. Blue: excitatory neurons, red: inhibitory neurons. **D-F** Spiking statistics for each population across all 32 areas simulated using 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 (**?**) averaged across neurons.