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 the dynamics and, ultimately, the function of the brain. However, even small mammals such as mice have around 1×10^{12} synaptic connections (1), the strengths of which are typically modelled as individual floating point values. If single precision floating point were used to store these values, several terabytes of memory would be required. As such memory requirements are beyond what is plausible for a single machine, simulations of large-scale spiking neural network currently are typically simulated on large distributed systems. Large parts of such models are typically described by simple algorithms which describe connectivity and the strength of synaptic connections. In this work, we describe our extensions to GeNN (2) – our GPU-based spiking neural network simulator – to enable it to 'procedurally' generate connectivity and synaptic weights as spikes are received rather than retrieving them in memory. 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 under-utilised 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 present the results of simulations of 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 and show that the results are correct and the simulation runs faster than previous simulations which ran on over 1000 supercomputer nodes.

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

While the brain of a mouse has around 70×10^6 neurons, their numbers are dwarved by the 1×10^{12} synapses which connect them. Computationally, simulating spikes propagating 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 a simulation timestep.

Because of the high memory requirements of large-scale brain models, they are typically simulated on large distributed systems using software such as NEST (3) or NEURON (4). By careful design, such simulators can maintain a constant memory requirements for each node can kept constant even when a simulation is distributed across thousands of nodes (5). However, such systems are large, expensive and consume large amounts of power meaning that they are typically shared between many researchers from many institutions.

Neuromorphic systems (6–11) take inspiration from the brain and have been developed specifically for simulating spiking neural networks. One particular relevant feature of the brain is that its memory elements – the synapses – are located throughout the system rather than being centrally located. In neuromorphic systems, this often translates to a large proportion of each chip being dedicated 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 (11), 4 (6) or even 1 bit (7, 9). While such strategies allow some classes of spiking neural networks to be simulated very efficiently, in the context of large-scale brain simulation, reducing the degree of connectivity to fit within the constraints of such a system inevitably changes its dynamics (12). Unlike the majority of other other neuromorphic systems, SpiNNaker (8) is entirely programmable and combines a large amount of on-chip meory 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 (13). **(TODO: good argument against SpiNNaker)**

GPU architectures 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 meaning that, as long as there is sufficient computation to be performed, the latency of accessing external memory can be 'hidden' behind computation. For example, each CUDA core of a modern GPU needs to perform approximately 10 arithmetic operations per byte of data accessed from memory in order to successfully hide the memory latency. However, processing a synapse is likely to require accessing approximately 8 B of memory and performing many fewer than 80 instructions, making this operating highly

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

memory bound. Nonetheless, in our previous work (14) we showed 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. Nonetheless, 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 **(TODO: cite)** interconnect, beyond this scaling will be dictated by the same communication overheads as other distributed systems.

In this work we present an 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 the memory and memory bandwidth requirements and make large-scale brain simulations on a single workstation possible.

Results

In the following subsections we will first present two novel features of our GeNN simulator (2) which allow it be used for simulating large-scale models on a single GPU. Finally, we will demonstrate the power of both features by simulating a recent model of the Macaque visual cortex (15) consisting of 4.13×10^6 neurons and 24.2×10^9 synapses on a single GPU and demonstratint that, not only are the results correct, but the simulation runs faster than simulations previously run on a supercomputer system.

Procedural connectivity. Our GeNN simulator (2) 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 (14). 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 implementation. 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 at the expense of equally limited memory bandwidth. Instead, what if we used some of the

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 (16), but subsequently widely used as a scalable benchmark (17). 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 (18) – all other SNN simulators which use code generation to generate all of their simulation code (rather than, for example NESTML (19), 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 Fig. 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, Fig. 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. 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 further using the profiler showed that this drop in performance was accompanied by a large number of "No instruction" stalls (events preventing the GPU from doing any work during a given clock cycle) as illustrated in Fig. 2D.

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

NeuronUpdateGroup neuronUpdateGroup
{
  {1000, d_VA},
  {1000, d_VB},
```

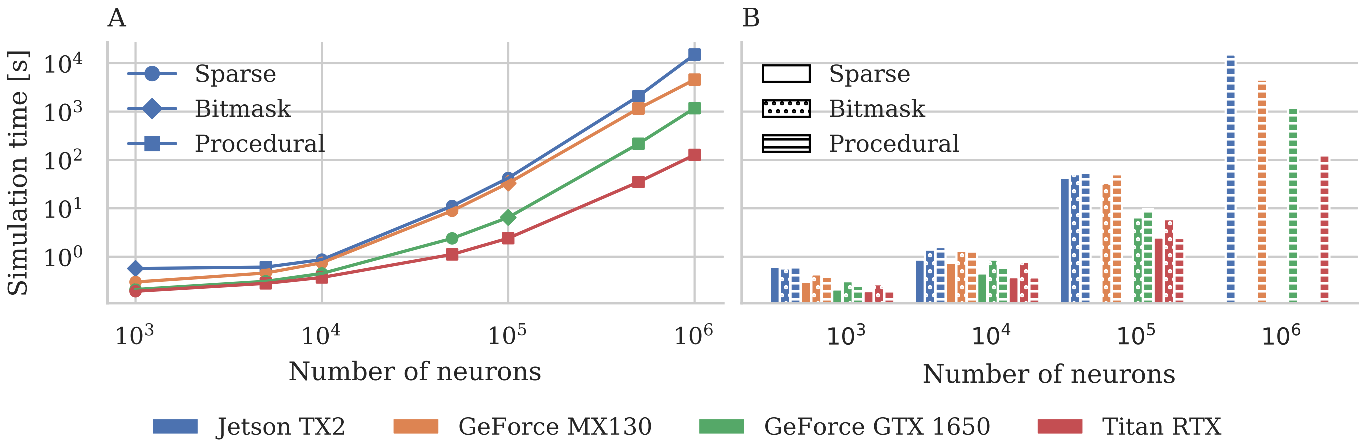

Fig. 1. Performance scaling on a range of modern GPUs. **A** The best performing approach at each scale. **B** Raw performance of each approach.

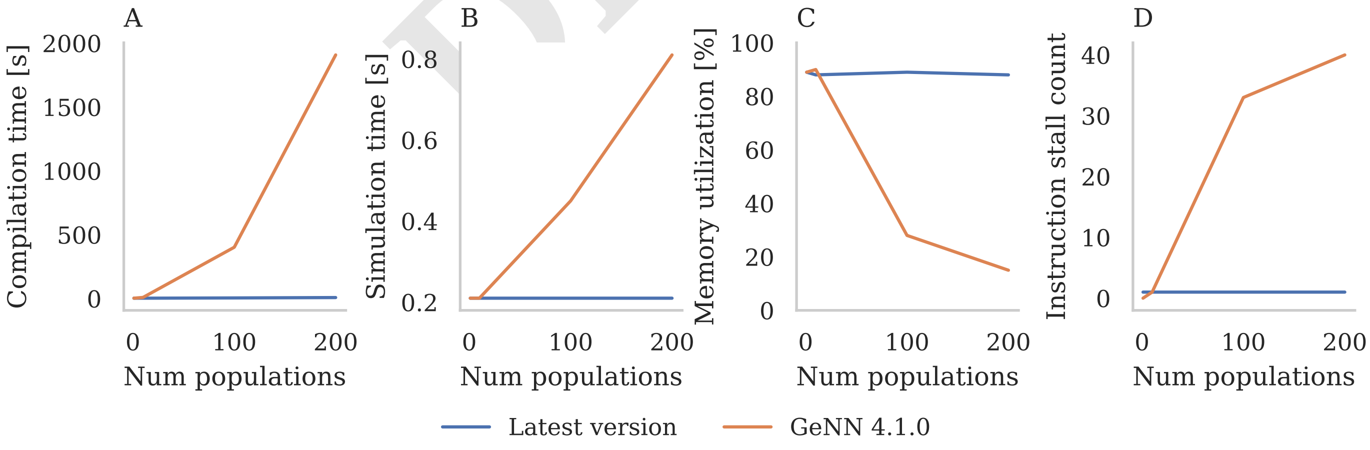

Fig. 2. Performance of a simulation of $1\,000\,000$ LIF neurons driven by a gaussian input current, partitioned into varying numbers of populations. **A** Compilation time using GCC 7.5.0. **B** Simulation time for an $1\,\mathrm{s}$ simulation. **C** Memory throughput reported by NVIDIA Nsight compute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

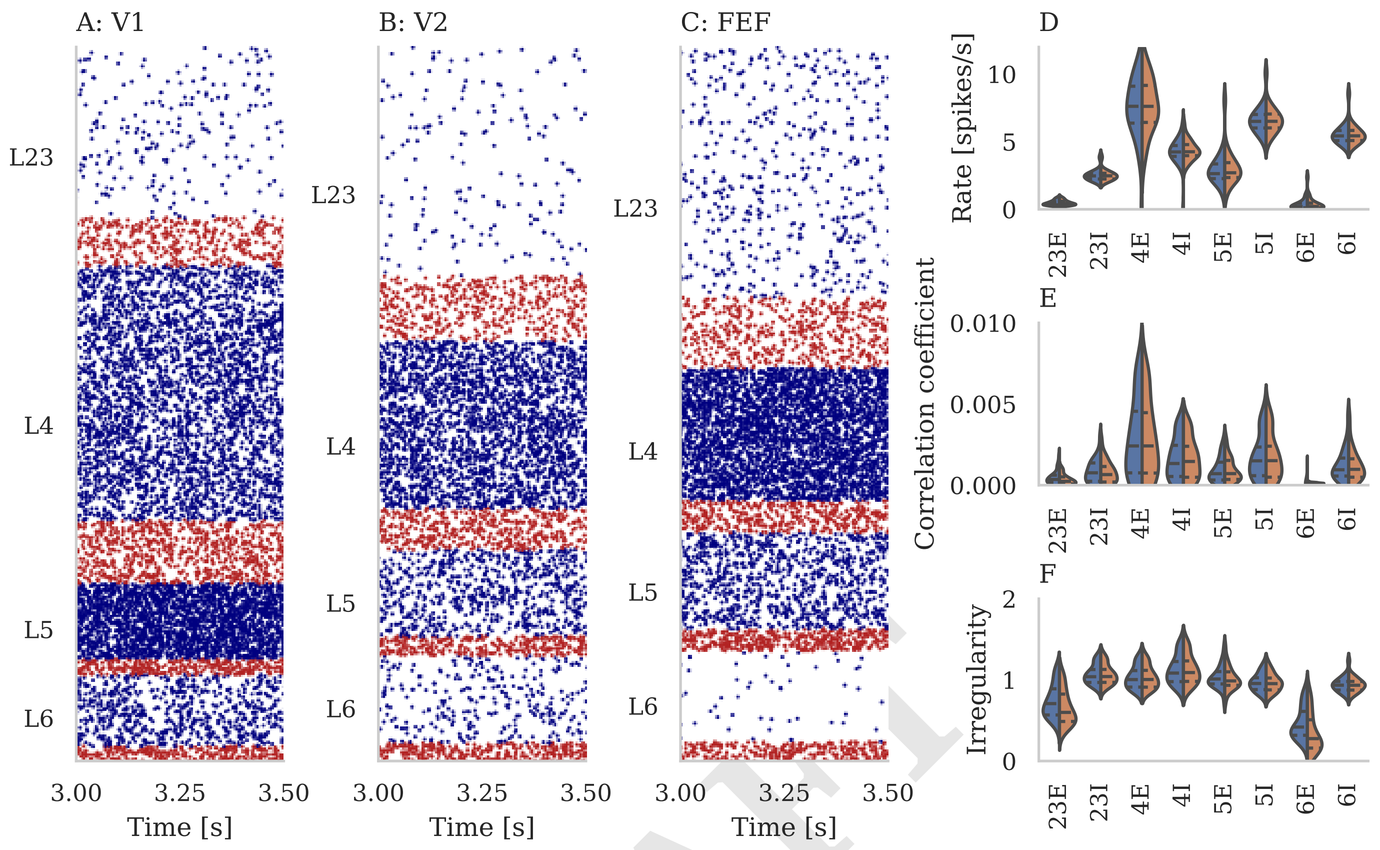

Fig. 3. Results of full-scale multi-area model simulation. **A-C** Raster plots of spiking activity of $3\,\%$ of the neurons in area V1 **A**, V2 **B**, and FEF **C**. Blue: excitatory neurons, red: inhibitory neurons. **D-F** Spiking statistics for each population across all 32 areas simulated using 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 (20) averaged across neurons.

```
  {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 (21, 22) 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 only be captured by modelling interconnected microcircuits at the level of individual cells. The multi-area model (15) does just this, using scaled versions of a previous 4 layer microcircuit model (22) to implement $1\,\text{mm}^2$ 'patches' for each of 32 areas of the macaque cortex involved in visual processing. The 32 areas are connected together with connectivity based on inter-area axon tracing data from the CoCoMac (23) database, further refined using additional anatomical data (24) and heuristics (25). These populations are connected toget

By using a supercomputer to simulate a model based on the latest connectivity data and The multi-scale model of the macaque visual cortex (15) 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.

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. Yavuz E, Turner J, Nowotny T (2016) GeNN: a code generation framework for accelerated brain simulations. *Scientific reports* 6(November 2015):18854.
3. Gewaltig MO, Diesmann M (2007) NEST (NEural Simulation Tool). *Scholarpedia* 2(4):1430.
4. Carnevale NT, Hines ML (2006) *The NEURON book.* (Cambridge University Press).
5. Jordan J, et al. (2018) Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. *Frontiers in Neuroinformatics* 12(February):2.
6. 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.
7. 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.
8. Furber SB, Galluppi F, Temple S, Plana LA (2014) The SpiNNaker Project. *Proceedings of the IEEE* 102(5):652–665.
9. Merolla PA, et al. (2014) A million spiking-neuron integrated circuit with a scalable communication network and interface. *{S}cience* 345(6197):668–673.
10. 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.
11. 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.
12. 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.
13. Rhodes O, et al. (2019) Real-Time Cortical Simulation on Neuromorphic Hardware.
14. 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.
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. 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.
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. Blundell I, et al. (2018) Code Generation in Computational Neuroscience: A Review of Tools and Techniques. *Frontiers in Neuroinformatics* 12(November).
19. Plotnikov D, et al. (2016) NESTML: a modeling language for spiking neurons. pp. 93–108.
20. Shinomoto S, et al. (2009) Relating neuronal firing patterns to functional differentiation of cerebral cortex. *PLoS Computational Biology* 5(7).
21. 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.
22. 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.
23. Bakker R, Wachtler T, Diesmann M (2012) CoCoMac 2.0 and the future of tract-tracing databases. *Frontiers in Neuroinformatics* 6(DEC):1–6.
24. (2014) A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cerebral Cortex* 24(1):17–36.
25. Ercsey-Ravasz M, et al. (2013) A Predictive Network Model of Cerebral Cortical Connectivity Based on a Distance Rule. *Neuron* 80(1):184–197.