Neuromorphic Computing and Engineering

PURPOSE-LED PUBLISHING™

PAPER • OPEN ACCESS

Scalable construction of spiking neural networks using up to thousands of GPUs

To cite this article: Bruno Golosio *et al* 2026 *Neuromorph. Comput. Eng.* **6** 024012

View the article online for updates and enhancements.

You may also like

- Sub-realtime simulation of a neuronal network of natural density
Anno C Kurth, Johanna Senk, Dennis Terhorst et al.

- Scaling neural simulations in STACS
Felix Wang, Shruti Kulkarni, Bradley Theilman et al.

- A Parallel Supercomputer Implementation of a Biological Inspired Neural Network and its use for Pattern Recognition
Vincent de Ladurantaye, Jean Lavoie, Jocelyn Bergeron et al.

This content was downloaded from IP address 192.167.140.21 on 14/09/2026 at 15:16

NEUROMORPHIC
Computing and Engineering

Check for updates

OPEN ACCESS

RECEIVED
5 February 2026

REVISED
9 April 2026

ACCEPTED FOR PUBLICATION
28 April 2026

PUBLISHED
14 May 2026

Original content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

Scalable construction of spiking neural networks using up to thousands of GPUs

Bruno Golosio[1,2,12], **Gianmarco Tiddia**[2,1,12,*], **José Villamar**[3,4,12], **Luca Pontisso**[5], **Luca Sergi**[1,2], **Francesco Simula**[5], **Pooja Babu**[6,7], **Elena Pastorelli**[5], **Abigail Morrison**[3,7], **Markus Diesmann**[3,8,9,10], **Alessandro Lonardo**[5], **Pier Stanislao Paolucci**[5] and **Johanna Senk**[3,11]

[1] Department of Physics, University of Cagliari, Monserrato, Italy
[2] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Cagliari, Monserrato, Italy
[3] Institute for Advanced Simulation (IAS-6), Jülich Research Centre, Jülich, Germany
[4] RWTH Aachen University, Aachen, Germany
[5] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Roma, Rome, Italy
[6] Simulation and Data Lab Neuroscience, Jülich Supercomputer Centre, Jülich Research Centre, Jülich, Germany
[7] Department of Computer Science 3 - Software Engineering, RWTH Aachen University, Aachen, Germany
[8] JARA-Institute Brain Structure-Function Relationships (INM-10), Jülich Research Centre, Jülich, Germany
[9] Department of Psychiatry, Psychotherapy and Psychosomatics, School of Medicine, RWTH Aachen University, Aachen, Germany
[10] Department of Physics, Faculty 1, RWTH Aachen University, Aachen, Germany
[11] Sussex AI, School of Engineering and Informatics, University of Sussex, Brighton, United Kingdom
[12] These authors contributed equally to this work.
[*] Author to whom any correspondence should be addressed.

E-mail: gianmarco.tiddia@ca.infn.it

Keywords: spiking neural networks, large-scale simulations, GPU, exascale

Abstract

Diverse scientific and engineering research areas deal with discrete, time-stamped changes in large systems of interacting delay differential equations. Simulating such complex systems at scale on high-performance computing clusters demands efficient management of communication and memory. Inspired by the human cerebral cortex—a sparsely connected network of $\mathcal{O}(10^{10})$ neurons, each forming $\mathcal{O}(10^3)$–$\mathcal{O}(10^4)$ synapses and communicating via short electrical pulses called spikes—we study the simulation of large-scale spiking neural networks for computational neuroscience research. This work presents a novel network construction method for multi-GPU clusters and upcoming exascale supercomputers using the message passing interface, where each process builds its local connectivity and prepares the data structures for efficient spike exchange across the cluster during state propagation. We demonstrate scaling performance of two cortical models using point-to-point and collective communication, respectively.

1. Introduction

Each one of the billions of cortical neurons sends up to around ten action potentials (spikes) per second with sub-millisecond temporal precision to thousands of other neurons through a network with intricate, hierarchical connectivity structure. Therefore, each cortical microcircuit below 1 mm^2 of brain surface can produce one billion synaptic events per second of biological activity [1]. This rough estimate points to two main challenges posed by neuroscience research on simulation technology: (1) brain-scale simulations at the level of individual neurons and synapses require substantial amounts of computer memory, (2) resolving the neuronal and synaptic dynamics with sufficient accuracy even in long simulations requires high processing speeds [2]. High-performance computing (HPC) provides high-end hardware in pre-exascale and exascale supercomputers. Drawing their compute power from thousands of general-purpose GPUs connected with InfiniBand or other high-speed networks [3], these systems are promising candidates for approaching the required scales and speeds. To harness these distributed systems effectively, simulation software with parallelized algorithms and efficient information exchange between compute nodes needs to be developed and optimized. The message passing interface (MPI) [4]

© 2026 The Author(s). Published by IOP Publishing Ltd

is the main standard for inter-node communication, offering both point-to-point and collective communication between groups of processes.

So far, no GPU-based simulation software for point-neuron spiking neural networks (SNNs) has been reported that scales up to entire compute clusters, fulfilling the infrastructure and accuracy demands of computational neuroscience. Alternative approaches that achieve a subset of these goals include the Digital Brain [5, 6], which simulates networks with up to 10^{10} neurons on thousands of GPUs using MPI, but without storing individual synapse information. Similarly, GeNN [7] simulates millions of neurons on a single GPU using an on-the-fly connectivity generation method [8]. CARLsim supports multi-GPU execution on a single machine [9, 10]. NEURON [11], enhanced by CoreNEURON, already combines MPI with GPU acceleration with a focus on multi-compartment neuron models [12, 13]. They demonstrate the parallel evaluation of a large number of differential equations required to update the complex neuron dynamics. In contrast, the main bottleneck in distributed simulations of large networks of point neurons is the spike communication between different nodes of a compute cluster, and this holds for nodes equipped with GPUs or CPUs [14, 15]. STACS [16] tried to overcome this bottleneck by leveraging Charm++ parallelization capabilities, showing good scaling performance in the simulation of balanced network models [17].

The simulation code NEST [18] employs a hybrid approach using MPI for inter-node and OpenMP [19] for intra-node parallelization on modern multi-core processors (CPUs). Benchmark SNN models have reached up to 10^9 neurons and 10^{13} synapses [20, 21], while models representing distinct brain regions with millions of neurons and billions of synapses have been simulated using thousands of cores in parallel [22–26]. These efforts have made NEST a reference platform for large-scale SNN simulations [1], keeping pace with conventional hardware development and the increasing interest in scalable neuromorphic systems [27]. The round-robin algorithm to distribute neurons on MPI processes and the collective communication [21] for synchronous spike exchange between processes used by NEST is optimal for load balancing assuming homogeneous network structure and neuron activity. However, this approach does not exploit the heterogeneity of biological brains, including their topological and spatial multi-scale network organization and transient changes in spiking statistics [28].

The recent GPU implementation of NEST addresses these demands. NEST GPU [29, 30] supports multi-GPU simulations and takes advantage of the locality of neuron populations. Like the CPU version, the C++ code exposes a handful of functions to the Python level for the setup and execution of concrete network models, with some of them being validated against the CPU version [15, 29]. The code uses CUDA to interface GPU hardware and MPI for point-to-point communication between compute nodes [15]. The initial algorithm for network construction relied on a transfer from CPU to GPU memory. To reduce computation times, [30] proposed a method to dynamically build the network directly in GPU memory, applicable to single-GPU simulations.

Going beyond these previous works, here we present memory-efficient data structures for in-GPU network construction and spike communication during simulation on multi-GPU systems. Each MPI process builds its part of the network without MPI communication with other processes, creates the data structures needed to communicate via MPI, and efficiently transmits spikes across remote connections using either point-to-point or collective communication according to the user's choice. As demonstrated by a reference implementation, the new method enables large-scale SNN simulations of up to thousands of GPUs exploiting a computational communication scheme inspired by the temporal and spatial sparsity observed in the brain. Our results indicate that network sizes of 2×10^{10} neurons and 10^{14} synapses can be reached with our approach on the upcoming exascale supercomputer JUPITER [31].

2. Results

Simulations of neuronal networks typically consist of two main phases: *construction* and *simulation* (or *state propagation*). The former comprises all the steps needed before a loop advancing the dynamical state of the network begins. In our reference implementation, the network construction can be conceptually divided into subtasks such as the initialization phase, which includes the creation of neurons, devices (for stimulation or recording), and connections, and a phase that organizes data structures for spike delivery (here named *simulation preparation*); see also section 4.3 for details on the phases.

The main result of this work is the design of a novel algorithm for network construction that instantiates and organizes contiguous communication maps in GPU memory for efficient spike routing and

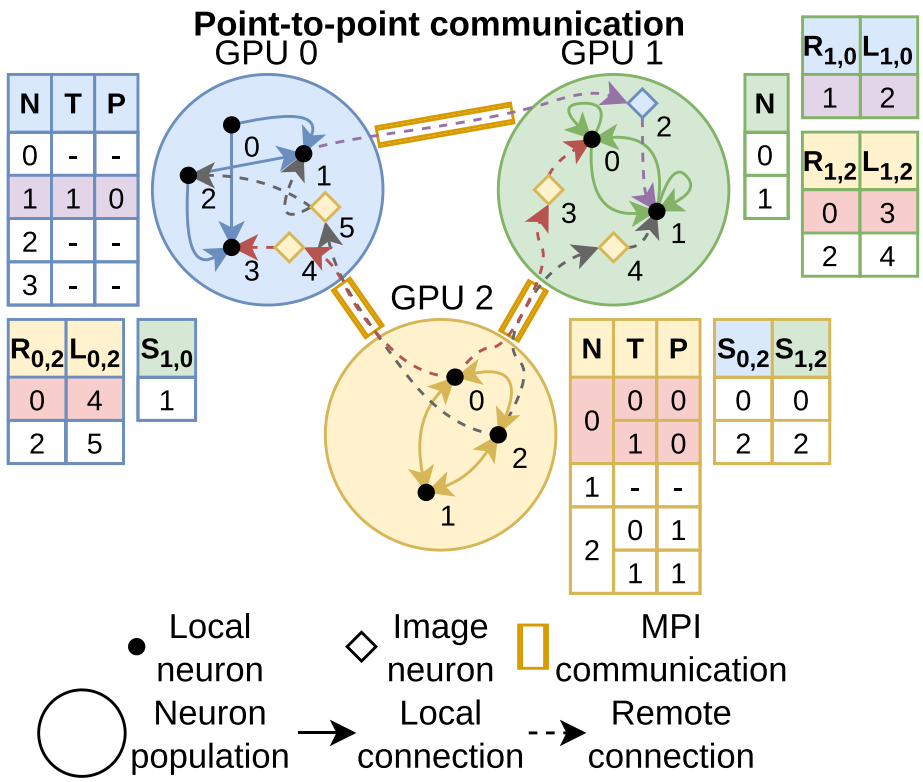

Figure 1. Communication scheme for point-to-point spike routing and delivery with one MPI process per GPU. Each process is identified by rank and color (0: blue, 1: green, 2: yellow). After neuron creation, all ranks contain an arbitrary number of neurons, each with a unique index in the range from 0 to $M_\sigma - 1$, where M_σ is the number of neurons in the rank σ. The $(\mathbf{R}_{\tau,\sigma}, \mathbf{L}_{\tau,\sigma})$ maps associate the index of the source neuron in rank σ to the index of its image in the target rank τ. The (**N**, **T**, **P**) tables associate to each local neuron, identified by the table row index **N**, the target processes **T** in which it has an image, and the corresponding position **P** of this image in the $(\mathbf{R}_{\tau,\sigma}, \mathbf{L}_{\tau,\sigma})$ maps. Spikes emitted by neuron 1 of rank 0 and neurons 0 and 2 of rank 2 are routed to their proxies in the corresponding target ranks, listed in array **T**, by sending the positions **P** in the maps $(\mathbf{R}_{\tau,\sigma}, \mathbf{L}_{\tau,\sigma})$ through MPI. $\mathbf{S}_{\tau,\sigma}$ in the source process corresponds to $\mathbf{R}_{\tau,\sigma}$ in the target process. Within each rank, remote connections from image neurons serve as the final link to local neurons receiving spikes. As an example (in red), when neuron 0 of rank 2 emits a spike, the (**N**, **T**, **P**) tables of that rank (bottom right, yellow heading) are used to retrieve the ranks of the target processes, **T**, in which this neuron has images (0 and 1 in this example), and the positions **P** of those images (0 in both cases) in the$(\mathbf{R}_{0,2}, \mathbf{L}_{0,2})$ and $(\mathbf{R}_{1,2}, \mathbf{L}_{1,2})$ maps, respectively. Those positions are sent through MPI by point-to-point communication (orange) and they are used by the target MPI processes (ranks 0 and 1) to retrieve the local indexes of the image nodes from the **L** columns of the maps. The spike is then treated as a local spike and is therefore sent to the target neurons through the outgoing connections of the image neuron. Similarly (in purple), spikes emitted by neuron 1 of rank 0 are routed to the image neuron 2 in rank 1. It can be observed that rank 1 has two (**R**, **L**) maps (top right, blue and yellow headings) because it has images of neurons from both 0 and 2 ranks.

delivery. Similar to the parallel network construction methods previously developed by Jordan *et al* [21], this algorithm relies on each MPI process independently building local connectivity and preparing data structures to manage spikes transmitted to neurons on other processes, avoiding communication between MPI processes during this phase. Here, we distinguish between local connections, linking neurons within the same MPI process, and remote connections, spanning neurons across different processes. Our method uses proxy representations of neurons for remote connections, which are an image of source neurons in the target processes. With each MPI process handling a single GPU, connections can then be generated locally in GPU memory following the approach described in [30]. Figures 1 and 2 illustrate how our communication maps are used for spike routing and delivery and further details can be found in section 4.1. The mapping structures support both point-to-point and collective MPI communication, providing flexibility to adapt to network architecture and activity patterns. The following subsections elaborate on these two communication schemes and show performance results.

2.1. Point-to-point communication: multi-area model (MAM)

The MAM of 32 vision-related areas of macaque monkey cortex [22, 32] serves as a neuroscientifically relevant model to assess simulator performance in a network with complex, hierarchical architecture and biologically plausible spike statistics. The model represents each area by a 1 mm^2 patch with the full thickness of the cortex and natural density of neurons and synapses, comprising a total of 4.13×10^6 neurons and 24.2×10^9 synapses. See section 4.2.1 for additional details on the model. Simulations are

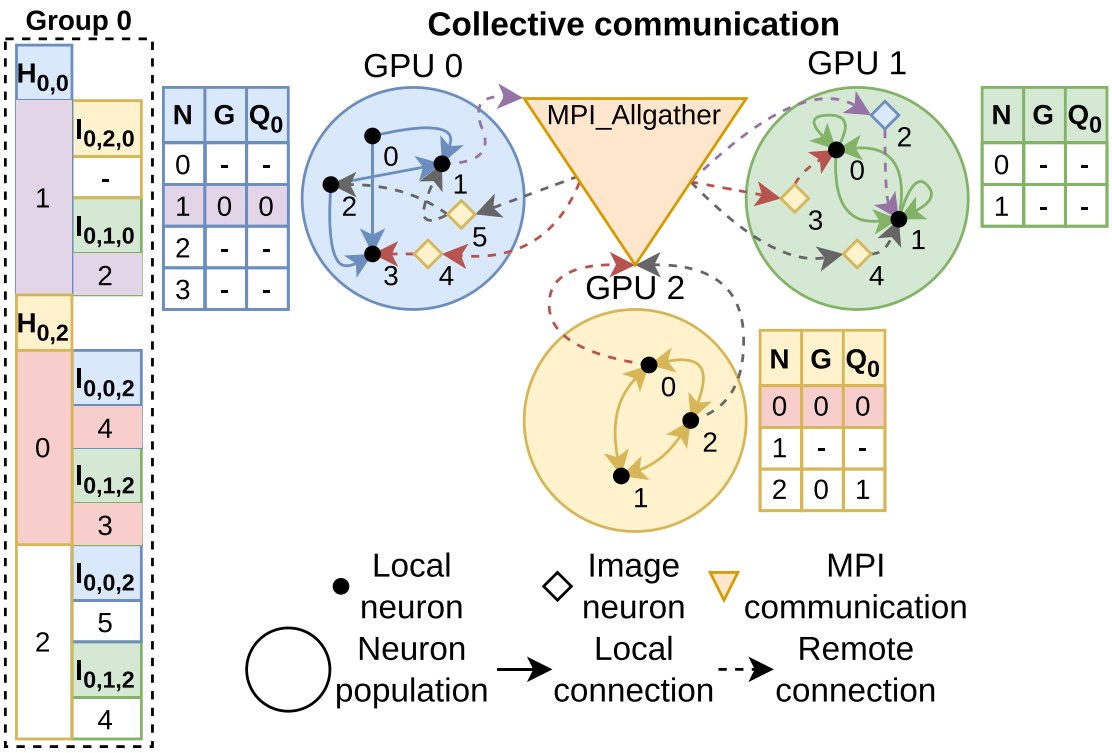

Figure 2. Communication scheme for collective spike routing and delivery with one MPI process per GPU. Each process is identified by rank and color (0: blue, 1: green, 2: yellow). After neuron creation, all ranks contain an arbitrary number of neurons, each with a unique index in the rank local neuron array **N**. All ranks belonging to an MPI group use specific indexing arrays mirrored across all members of the group, here group 0 uses arrays denoted with suffix 0. Neurons of rank σ, which have a proxy in any other rank of the group, are indexed in the host array, here $\mathbf{H}_{0,\sigma}$. Spikes emitted by neurons 1 of rank 0 and neurons 0 and 2 of rank 2 are first routed to the groups to which they have remote connections using the group array **G**, simultaneous participation to multiple groups is possible; in this illustration only group 0 exits. $\mathbf{Q}_0$ stores the positions of source neuron indexes in the host array, these are used to exchange the indexes of spiking neurons by all members of the group, and once received, spikes are routed to the image neurons using image index array $\mathbf{I}_{0,\tau,\sigma}$. Within each rank, remote connections from image neurons serve as the final link to local neurons receiving spikes. As an example (in red), when neuron 0 of rank 2 emits a spike, the (**N**, **G**, $\mathbf{Q}_0$) tables of that rank (bottom right, yellow heading) are used to retrieve the communication groups from **G** in which this neuron has images (only group 0 this example), and its position in the host specific source index table $\mathbf{H}_{0,2}$ from table $\mathbf{Q}_0$ (0 in this example). Its position is sent through MPI by collective communication (orange) and is received by both ranks 0 and 1. Rank 0 uses the map $\mathbf{I}_{0,0,2}$ to retrieve the local index of the image node, and treats the incoming spike spike as a local spike, therefore sending the spike to the target neurons through the outgoing connections of the image neuron. In the same way, rank 1 uses the map $\mathbf{I}_{0,1,2}$ to retrieve the local index of the image node to route the spike to local targets. Similarly (in purple), spikes emitted by neuron 1 of rank 0 are routed to the image neuron 2 in rank 1.

performed in the model's *metastable state*, reflecting resting-state activity in lightly anesthetized monkeys. We use the MPI point-to-point communication protocol for this model. *Point-to-point* protocols enable direct communication between two processes and are advantageous for networks with uneven distribution of neurons or synapses, resulting in highly heterogeneous communication patterns between processes.

We previously used this model to explore the potential of multi-GPU systems for speeding up the state propagation phase [15]. In that work, the network construction phase is carried out on the CPU. As in [30], we call this algorithm *offboard*, to distinguish it from the new *onboard* version that performs this phase directly on the GPU. Moreover, in [15] we compared the results of the spike statistics with the CPU version of the NEST simulator. Similarly to what has been done in that work, we here validated the results of the spike statistics of the model with respect to the previous version of the simulation code. Despite the high degree of variability in the MAM metastable state [22], the validation, introduced in section 4.4 and presented in detail in appendix A, underlines the compatibility between the two versions.

Here, simulations follow the same configuration as [15] for the *offboard* version, using 32 NVIDIA V100 GPUs on the JUSUF cluster [33], with one GPU per cortical area due to the memory restriction of the GPUs. In comparison with [15], the present work shows the performance improvements obtained using the new algorithm for network construction. Additionally, a slight simulation performance gain can be noticed thanks to general optimizations to our state propagation implementation.

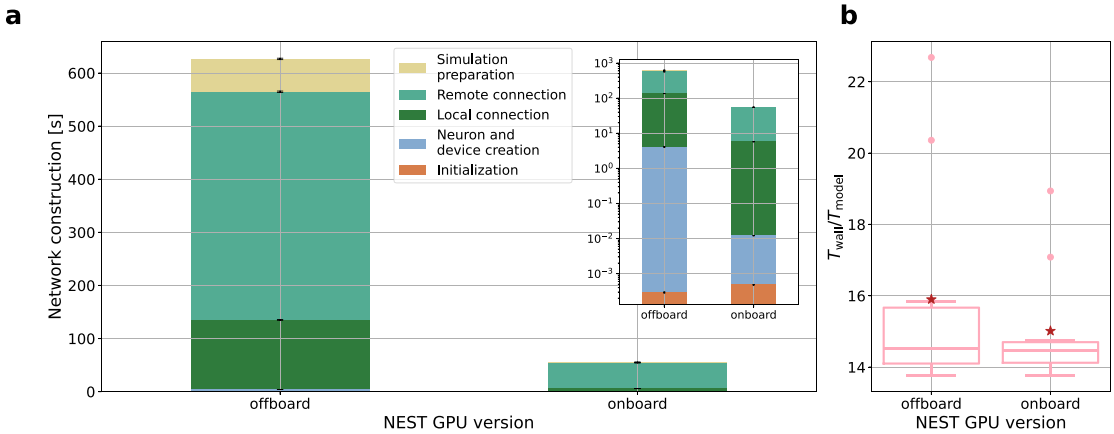

Figure 3. Comparison between performance of *offboard* and *onboard* versions on the simulation of the MAM in the metastable state. Simulations are performed on 32 nodes on JUSUF (one NVIDIA V100 GPU per node). Mean data shown from averaging over 10 simulations using different random seeds. Black error bars represent the standard deviations, which are barely visible due to the little variation across different simulations. (a) Bar plot of the network construction time divided into its subtasks, which are shown in chronological order from bottom to top: (1) simulator initialization, (2) neuron and device creation, (3) local connection generation, (4) remote connection generation, and (5) the organization of data structures for spike delivery (*simulation preparation*). Bar heights represent mean values, and black error bars standard deviations. At the top-right of the panel, an inset shows the phases of the network construction on a logarithmic scale to better visualize the contribution of initialization and neuron and device creation phases. (b) Box plot of state propagation time measured as the real-time factor. The central line represents the median, whereas the box indicates the interquartile range (IQR). The whiskers extend up to 1.5 ×IQR, and data exceeding this value are represented as outliers. Red stars indicate the mean of the distributions.

Figure 3 compares *offboard* and *onboard* in the simulation of the MAM. Network construction time shown in panel **a** is more than 10 times faster with the *onboard* version. In particular, network construction is 686.0 ± 1.5 s for the *offboard* and 55.5 ± 0.1 s for the *onboard* version. This time is divided into the subtasks of network construction. Connections creation, i.e. the most time-consuming phases of the network construction, shows 20 × and 9 × speedup for the *onboard* version for local and remote connections creation, respectively. Neuron and device creation and simulation preparation, which have only a small contribution to the total network construction time also in the case of the *offboard* version, are the phases benefiting from the largest speedup, respectively, of 350 times and 50 times. The state propagation time, represented in panel **b** with the real-time factor (RTF) (see equation (21)), is comparable between the two versions: 16.0 ± 3.0 and 15.0 ± 1.7 for *offboard* and *onboard* versions. The corresponding median values are 14.5 and 14.4, respectively. The differences in the extent of the two boxes and the outliers are related to the activity variability of the metastable state of the model, which may result in fluctuations in the spiking activity.

On the JUSUF cluster, each NVIDIA V100 GPU accommodates only a single MAM area. The NVIDIA A100 GPUs with 64 GB of GPU memory of the Leonardo Booster supercomputer [3], however, may host multiple areas on a single GPU card (see also section 4.5). An *area packing* algorithm has been specifically designed for the NEST GPU implementation of the MAM model, which enables the simulation of the model using down to 8 GPUs. These results are described in appendix B, and the algorithm is described in section 4.2.1.

2.2. Collective communication: scalable balanced network

Weak scaling performance is assessed with a two-population random balanced network of excitatory and inhibitory neurons [34], similar to the one used in previous scaling studies [28]. Using random, fixed in-degree connectivity (multapses and autapses allowed), in principle, any neuron could be connected to any other one in that network. The average firing rate of excitatory and inhibitory populations is around 8 spikes per second. Section 4.2.2 provides further details on that model. The total network size is proportional to the product of a scale factor for the number of neurons per MPI process (`scale` parameter) and the total number of MPI processes. The scale is constrained by the GPU memory available per MPI process.

We run simulations on the Leonardo Booster supercomputer [3], which is equipped with four NVIDIA A100 GPUs per cluster node, using one MPI process per GPU device, and employing collective MPI communication. *Collective protocols* allow the distribution of a communication payload across multiple processes. When network load across nodes is balanced and communication payloads are homogeneous, collective protocols provide latency, bandwidth, and overall communication speed benefits over

Table 1. Scalable balanced network model size (i.e. total number of neurons and synapses) as a function of the number of compute nodes. Each node is equipped with four GPUs, and the `scale` of the model is set to 20. The neuron number at `scale`= 1 is 11 250; the in-degree is fixed to 11 250.

Nodes	GPUs	Neurons ($\times 10^6$)	Synapses ($\times 10^{12}$)
32	128	28.8	0.32
64	256	57.6	0.65
96	384	86.4	0.97
128	512	115.2	1.30
192	768	172.8	1.94
256	1024	230.4	2.59

point-to-point methods. Results presented here correspond to `scale` = 20 (i.e. 2.25×10^5 neurons and 2.53×10^9 synapses per GPU), which is the scale that would virtually enable the execution of such a model on the full Leonardo Booster. Table 1 shows the model size in terms of number of neurons and synapses as a function of the number of nodes and GPUs used in a simulation. Indeed, other choices are possible: appendix C reports node creation and connection time, as well as simulation preparation times, for the network model simulated at scales 10 and 30.

We perform simulations using from 32 to 256 nodes, i.e. up to 1024 GPUs. As CINECA regulations limit larger runs, we provide estimates for configurations requiring more nodes. Since each MPI process carries out network construction and simulation preparation independently without the need for communication, it is possible to assess performance using a single compute node, with four MPI processes. For this, each process constructs its regular share of a large neuronal network in the absence of the remainder of the network. While no state propagation occurs, this approach estimates GPU memory usage and construction time for large configurations at lower costs. We refer to such results as *estimated* as opposed to *simulated* outcomes.

For clarity, when evaluating simulated results, we analyze data from all processes active during simulation. For instance, during simulations using 1024 processes, we analyze 1024 data samples of performance and memory usage metrics. On the contrary, estimating the outcome of 1024 processes using only 4 processes means we only analyze 4 data samples. In this case, each of the 4 processes performs a *dry-run* of network construction and simulation preparation as if 1024 processes were being used. When estimating the results of stochastic connection rules, the performance and memory usage data reported by the limited number of processes performing the estimation are only a few samples of the actual distribution across the larger estimated number of processes. Naturally, a single estimation run does not yield the full distribution of performance and memory usage. For statistical significance, we use 4 processes per estimation run and average over 5 estimation runs. To obtain a complete estimate there exists literature on HPC exploring statistical methods to predict memory and time requirements to run distributed applications [35]; however, these methodologies are beyond the scope of the present work.

In the following, we explore four alternative algorithms for balancing GPU memory usage and performance. Large-scale simulations occupy a relevant fraction of memory by data structures mapping remote source neurons to their local image neurons and outgoing connections on target MPI processes. To reduce GPU memory usage, parts of these structures can be stored in CPU memory, at the cost of slower network construction and simulation. The optimizations are ordered by increasing GPU memory usage, referred to as GPU memory levels (GMLs), from level 0 to level 3. A detailed description of the different levels can be found in the section 4.1.

Figure 4 summarizes network construction (panel a) and state propagation (panel b) time with increasing number of compute nodes for all GMLs as a function of the number of cluster nodes. As expected, higher GMLs yield faster simulation speeds as data movement between CPU and GPU for communication is reduced due to the increasing availability of communication maps in GPU memory. For the GML 3, which is the most efficient one in terms of time-to-solution, we also show the performance when spike recording is disabled, which show, on average a 20% reduction in state propagation time. Furthermore, network construction speed also benefits from higher GMLs as within each GPU, creation and sorting of the communication maps is performed in parallel. However, lower GMLs are not without any gain, the lower the optimization, the lower the memory footprint of simulation-related data structures in GPU.

Figure 5 shows the peak memory usage as a function of the number of compute nodes. Characterization of this peak is critical, as transient memory allocation can trigger out-of-memory errors

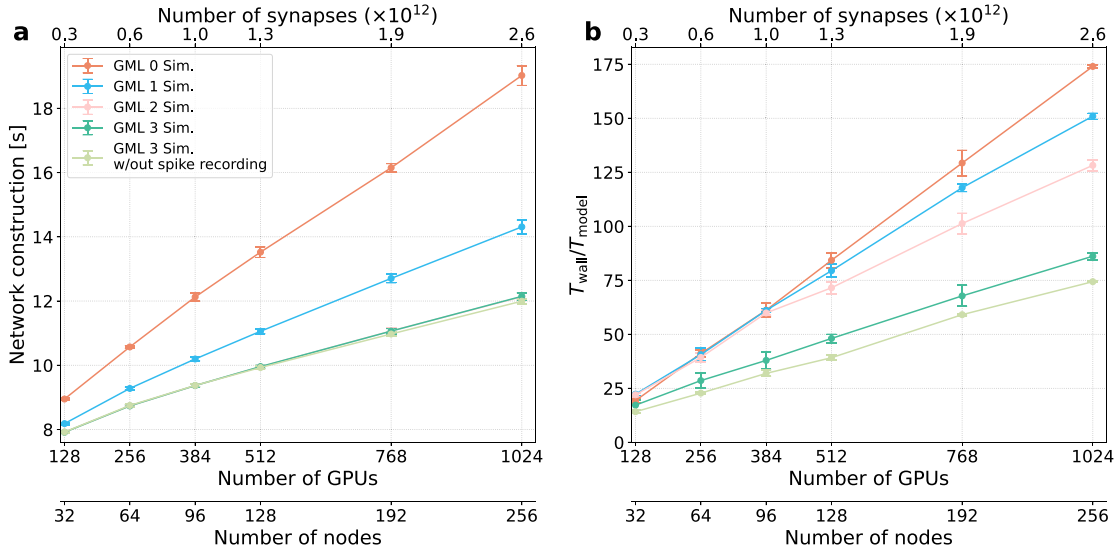

Figure 4. Network construction (a) and state propagation (b) times of the scalable balanced network model for the four GPU memory levels as a function of the number of cluster nodes (four GPUs per node). State propagation is measured as the real-time factor. The GPU memory level 3 was also reported when spike recording was disabled. GPU memory levels 2 and 3 in the network construction overlap.

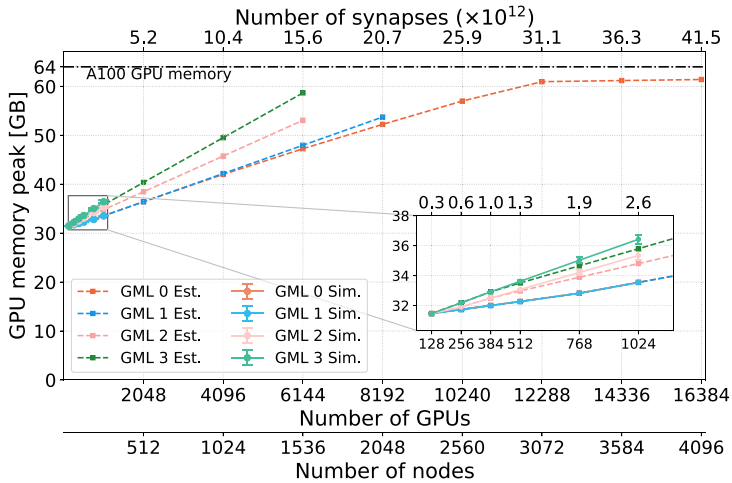

Figure 5. Peak memory usage per GPU for the scalable balanced network model simulation as a function of the number of cluster nodes for the four GPU memory levels. The upper horizontal axis shows the total number of synapses of the model, indicating the increasing network size. Square markers and dashed lines represent the estimation of the memory peak obtained using four MPI processes on a single node. Dot markers and continuous lines represent simulated configurations over five simulations using different random seeds. Error bars represent standard deviations. On the bottom right of the figure, an inset of the measured data is shown to facilitate comparison with the estimated values. Dash-dotted horizontal line represents the memory limit of one NVIDIA A100 GPU. For these configurations, the first two levels of GPU memory show compatible results, and the points overlap.

and thus define the scalability limit of the simulation. This illustrates the configurations that would succeed in constructing the network for each GML.

GML 0 can reach a 4096-node configuration without exceeding the GPU memory of the A100 GPUs. From 3072 nodes onward, the GPU-memory peak plateaus due to the fixed number of in-degree per neuron of the network across all configurations. Since the incoming connections are evenly distributed among MPI processes and GML 0 copies to the GPU only the data of the remote neurons connected with the local ones, as the number of processes surpasses the in-degrees, the size of neuron maps copied does not increase anymore. Other GMLs require additional GPU memory, and the additional allocation for instantiating the network reaches the GPU memory limit. In particular, the simulations with GMLs 2 and 3 exhibit higher GPU peak memory usage than the respective estimates and also show variability across MPI processes. This observation is explained by the non-deterministic allocation of temporary buffers needed for the network construction with the stochastic connectivity operations.

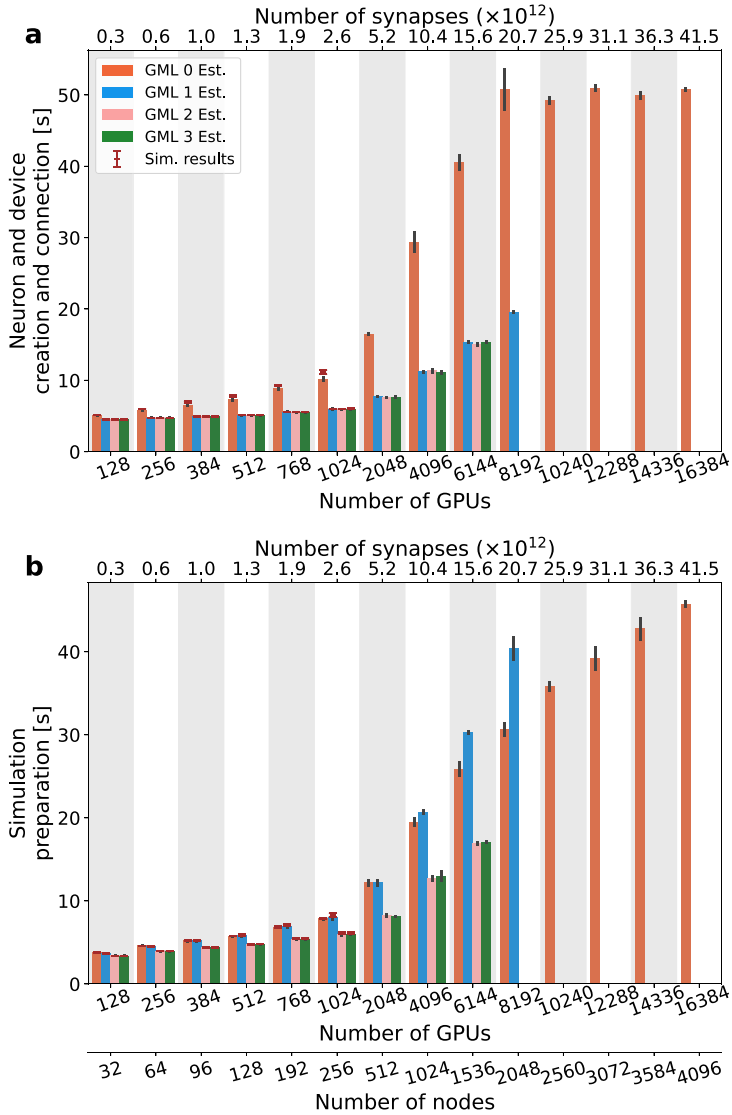

Figure 6. Network construction time divided into neuron creation and connection—panel (a) contribution of the neuron and device creation and the local and remote connection generation subtasks—and simulation preparation—panel (b) organization of data structures for spike delivery subtask—times of the scalable balanced network model evaluated across all MPI processes as a function of the number of cluster nodes for the three GPU memory levels. The bars represent an estimation using four MPI processes on a single node. The horizontal line markers show the time averaged across the MPI processes of five simulations with different seeds using the GPU memory levels. Errors for bars and horizontal line markers represent their standard deviation, with most of the values showing little variability.

Naturally, for each GML, we can also characterize the contribution of each network construction step to the overall time spent. Figure 6 splits network construction time into neuron and device creation, connection, and simulation preparation (organization of data structures for spike delivery) time as a function of the number of cluster nodes across all GMLs. Here, initialization time is not considered due to its negligible contribution to the overall time: as shown in figure 3(a), it is in the order of 10^{-2} s and does not scale with network size.

For node and device creation and connection, in figure 6(a), GML 0 has worse scaling compared to the higher GMLs, which profit from the parallelization of the instantiation of the communication maps.

Moreover, a slight divergence is shown between simulated and estimated values for GML 0, which increases with the number of nodes employed. On the one hand, the probabilistic connection rule leads to a variability across processes, similar to its effect discussed with respect to the GPU peak memory usage for higher GMLs. On the other hand, GML 0, using mostly the CPU for computing connections, is more susceptible to system jitter, which is commonly observed in large-scale systems [35]. Altogether, in such a large-scale environment, the cumulative effect of these sources of overhead on performance,

not captured by the performance estimation, may become visible. We provided more details on the differences between simulated and estimated results in appendix E.

Figure 6(b) shows a similar pattern for the simulation preparation step, with the exception of GML 1, which now shows a similar scaling behavior to GML 0. This is expected, since, as reported in section 4.1.6, these memory levels have similar approaches, but while for level 0 only the source neurons that have an image in the target MPI process are the ones that have a connection in the given MPI process, for level 1 all source neurons have an image. With this, while globally GML 1 is faster if the whole network construction time is considered (see figure 4(a)), the simulation preparation phase is slower than level 0 since the operations to organize the connectivity become slower when the map increases in size.

Clearly, both instantiating and sorting communication maps have a deep impact during network construction and the advantage of parallelizing these operations in GPU is noticeable across scales.

3. Discussion

The present study shows that GPU-based SNN simulations greatly benefit from optimizing memory and speed during network construction. We devised and implemented a scalable network construction method directly in the GPU, without the need for MPI communication during this phase. The method can balance GPU memory consumption and performance, supporting both point-to-point and collective MPI communication, two modes whose efficiency depend on model structure and strongly influence state propagation time. We report a more than ten times faster network construction of the large-scale MAM of the macaque vision-related cortex [22] compared to a previous implementation [15]. Furthermore, we demonstrate weak scaling of a balanced random network model simulation up to 1024 GPUs, corresponding to a network of 230.4×10^6 neurons and 2.59×10^{12} synapses, and provide a model to extrapolate performance beyond that.

At very-large model size, the GPU memory peak depends primarily on the number of connections allocated in each GPU, with other data structures, neuron parameters and spike recording having a negligible contribution. This is also true for neuron models with multiple receptors, already supported in NEST GPU, and would also be true for more complex neuron models, such as multicompartment neuron models, where one or more circular buffers are needed for each neuron compartment, similarly to what has already been done for the CPU version of NEST and the NEAT library [36]. Extrapolations suggest that the whole Leonardo Booster cluster (i.e. 3456 nodes) could simulate a model with $3.5 \cdot 10^{13}$ connections—approximately 10% of the connections in the human cortex–with an estimated node creation and connection time of around 50 s and a simulation preparation time of 42 s. However, such a network model is characterized by fully random connectivity. For a more realistic model of the cortex, more structured connectivity can further reduce simulation time if mapped efficiently onto the hardware. Indeed, several connections are lateral, with a decay length in the order of a few hundred micrometers [37]. Moreover, only a limited fraction of neurons project to remote areas [38]. From a simulation perspective, lateral and long-range connections could be handled differently: a hybrid scheme combining point-to-point communication for local spikes and collective MPI communication for long-range projections would be particularly advantageous. Going from the load balancing round-robin distribution of neurons to network-structure-aware model representations on the hardware requires advanced mapping strategies. In our current implementation, load balancing needs to be handled by the user, which is sub-optimal given the complexity of the physical hardware networking topology unique to each compute cluster, and could be addressed in the future. Moreover, the *area packing* algorithm developed here for the MAM implementation could be generalized to further models with modular structure. Most of such models are driven by anatomical and physiological data that are increasingly available and improving in resolution. In particular, human brain atlases and parcellations spanning tens to hundreds, up to around a thousand parcels, are commonly used together with inter-parcel connectivity matrices and with inter-parcel distance information (e.g. tract length or signal/conduction delay). These features can be leveraged by partitioning algorithms to optimize the distribution of the workload across processes [39, 40]. Furthermore, there exist SNN simulator implementations that employ graph partitioning algorithms to balance the compute load for distributed simulations [17, 41]. These could serve as algorithmic references and points of comparison for future load balancing implementations in NEST GPU, however such studies would require accompanying validation of spiking statistics across simulations to verify that network dynamics match.

Further challenges related to network connectivity include, for example, dynamic connectivity changes and loading of existing connectomes: first, modeling structural plasticity requires the efficient

creation and removal of synapses during the simulation at runtime. A flexible framework for structural plasticity on a single GPU has recently been published [42]. Algorithms for distributed simulations across multiple GPUs are subject to future work. Second, large-scale models with data-driven connectivity (e.g. [25]) would benefit from efficient routines for loading complete connectomes from file instead of the current practice of initiating the network graph every time prior to the simulation phase. Future work introducing support for the SONATA format [43], together with strategies for storing and subsequent reloading of network connectivity could enhance network construction capabilities accordingly.

Aside from those challenges, this work shows the capability for NEST GPU to meet large-scale simulation requirements. Memory footprints shown in figure 5 suggest that our reference implementation can scale to the largest current HPC systems. Emerging exascale machines are composed of fewer than ten thousand nodes, with each node providing high computational density. It could then host full machine runs and increase the neural network density at each node. The exascale supercomputer JUPITER Booster [31] uses NVIDIA GH200 super-chips (https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip, last visited: 05.09.2025). With this hardware reference, simulations of the scalable balanced network model would become possible at higher scale and almost double the number of compute nodes with respect to the Leonardo Booster, leading to simulations of networks with the same order of magnitude of neurons and synapses of the human cortex, while allocating individual synapse information.

4. Methods

4.1. Remote connections

The MPI protocol allows for the management of simulations running on multi-GPU systems and on GPU clusters in a conceptually similar way. In both cases, the communication of spikes and other types of data between GPU devices is handled via MPI, regardless of whether the devices belong to the same node or to different ones in the cluster. Here, we use the term *multi-GPU simulation* to refer to any simulation involving more than one GPU, regardless of their actual physical location.

In such simulations, it is useful to distinguish between *local* connections, in which both the source and target neurons are located in the same MPI process, and *remote* connections, which span different MPI processes. Local connections are created by the invocation of the `Connect` method [30], while remote connections are generated through a dedicated `RemoteConnect` method, which will be described in this section. The strategy used in this work for managing remote connections involves representing the source neuron in the target MPI process using a virtual *image neuron* (or *proxy*). Each local neuron can be uniquely indexed within an MPI rank σ, for the following definitions, the set $\mathbf{N}_\sigma = \{0, \ldots, M_\sigma - 1\}$, where M_σ is the number of neurons in the rank, $\mathbf{N}$ is the collection of these indexes, $\mathbf{N}_i$ is a neuron index, and the cardinality $|\mathbf{N}_\sigma| = M_\sigma$. Image neurons are not explicitly instantiated in memory but are identified by separate local indexes in the target MPI process.

This representation allows for remote connections to be built in the target process using the same mechanisms as local connections, with the local index of the image neuron serving as the source. Efficient management of remote connection creation and spike communication relies on mapping structures that associate the index of the remote source neuron in the source MPI process to the corresponding local index of the image neuron in the target MPI process, where the connection is allocated.

The details of how those maps are created and used are slightly different depending on which MPI communication approach is used, i.e. *point-to-point* or *collective*. In the following paragraphs, we will describe them, analyze their differences and discuss their advantages and disadvantages.

4.1.1. Point-to-point MPI communication

This approach is particularly useful when each source neuron of a population allocated in a certain source MPI process has a large number of connections directed to the same target MPI process. For example, this could be the case in models where the distribution of neurons among MPI processes takes spatial location into account, for connections between MPI processes representing adjacent brain regions.

Maps associate the source neurons of remote connections with their images in the target MPI process, defined as follows:

- In the target MPI process, there is one map for each possible source MPI process. Those maps are pairs of the type $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ where $R_{\tau,\sigma,i}$ is the remote source neuron index, i.e. the index of the source neuron in the MPI process where it is allocated, and $L_{\tau,\sigma,i}$ is the local image neuron index, i.e. the index of the image neuron in the target MPI process where the connection is stored. τ is the

target MPI process rank, σ is the source MPI process rank, and $i \in \{0, \ldots, N_{\tau,\sigma} - 1\}$ is the index that represents the position in the map ($N_{\tau,\sigma}$ being the number of elements in this map). The pairs are stored on each target MPI process using two arrays per source MPI process. These arrays are organized in fixed-size blocks that are allocated dynamically in order to use GPU memory efficiently. Note that the arrays of index τ are allocated only in the target MPI process of rank τ.

- In the source MPI process, there is one sequence for each possible target MPI process, $S_{\tau,\sigma,i}$, where τ, σ and i follow the previous definitions and $S_{\tau,\sigma,1}$ is the local source neuron index.

Clearly,

$$S_{\tau,\sigma,i} = R_{\tau,\sigma,i}; \tag{1}$$

however, the first array is stored in the source MPI process (rank σ), while the second is stored in the target MPI process (rank τ). In the following text, we will use boldface characters to represent sequences such as:

$$\mathbf{R}_{\tau,\sigma} = \left(R_{\tau,\sigma,0}, \ldots, R_{\tau,\sigma,N_{\tau,\sigma}-1}\right) \tag{2}$$

with similar definitions for $\mathbf{L}_{\tau,\sigma}$ and $\mathbf{S}_{\tau,\sigma}$.

Map elements are sorted in ascending order with respect to the source neuron index, i.e.

$$S_{\tau,\sigma,i} \geqslant S_{\tau,\sigma,i-1} \quad \forall i \in \{1, \ldots, N_{\tau,\sigma} - 1\}. \tag{3}$$

Additionally, for each neuron allocated in the MPI process σ, i.e. for each index $s = 0, \ldots, M_\sigma - 1$, where M_σ is the number of neurons in σ, we create the sequence $T_{\sigma,s,j}$, of the MPI processes in which s has an image, where $j = 0, \ldots, J_{\sigma,s} - 1$ is the index of the elements in the sequence and $J_{\sigma,s}$ is their number, i.e. the number of MPI processes on which s has an image. This sequence is combined with the positions $P_{\sigma,s,j}$ of the pairs (source-neuron index, image-neuron index) in the (R, L) maps. Specifically, each element $P_{\sigma,s,j}$ stores the index corresponding to the position of the pair within the (R, L) map in the target MPI process. The MPI processes $T_{\sigma,s,j}$ are the ones that need to be properly informed when the node s emits a spike. Figure 1 illustrates how the sequences $\mathbf{T}_{\sigma,s}$, $\mathbf{P}_{\sigma,s}$ and $\mathbf{L}_{\tau,\sigma}$ are used to communicate the spikes and to transmit them across the remote connections. Appendix F shows how the maps $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ and the sequences $\mathbf{L}_{\tau,\sigma}$, $\mathbf{T}_{\sigma,s}$ and $\mathbf{P}_{\sigma,s}$ are organized in GPU memory, and describes in a more detailed example how those structures are used to communicate remote spikes.

It should be noted that in the source MPI process we store the position of the image neuron in the map, rather than its index in the target MPI process. In fact, as explained below, this position can be determined during network construction without communication between processes, and this allows to considerably reduce the network creation time. The sequences $\mathbf{S}_{\tau,\sigma}$, stored in the source MPI process σ, must be aligned with the sequences $\mathbf{R}_{\tau,\sigma}$ and $\mathbf{L}_{\tau,\sigma}$ stored in the target MPI process τ (equation (1)). A difficulty in keeping this alignment arises from the fact that the maps are updated dynamically every time a remote connect function is instantiated, and in many connection rules, the source and target neurons of the connections are determined from probabilistic distributions using pseudo-random numbers. A possible solution could be to create the maps only in the target MPI processes, and at the end of the network construction phase, just before the simulation of the network dynamics, to communicate them to the source MPI processes. However, for large networks, this communication has a deep impact on the network construction time [15]. The approach used in this work relies on a source process variant of the `RemoteConnect` method that is instantiated in the source MPI process, while the real method is used in the target process to create the connections. The method variant's sole function is to generate in the source MPI process the same source neuron indexes of the new connections generated in the target MPI process, based on the specified connection rule. These indexes are used to update the $\mathbf{S}_{\sigma,\tau}$ sequence in the source process and the $\mathbf{R}_{\tau,\sigma}$ and $\mathbf{L}_{\tau,\sigma}$ sequences on the target process each time the `RemoteConnect` method is instantiated. Generating the same source neuron indexes allows those sequences to be aligned (equation (1)) without the need for MPI communication during the network construction phase. Since some connection rules [44] require that the indexes of the source neurons be extracted using pseudo-random numbers according to given probabilistic rules, an array of generators, $\mathtt{RNG}_{\sigma,\tau}$, is used to generate these sequences. The $\mathtt{RNG}_{\sigma,\tau}$ generator starts from the same seed in the source MPI process σ and in the target MPI process τ, and is used exclusively to generate the indexes of the source neurons of the remote connections, so that these sequences are always aligned between the two processes.

4.1.2. Collective MPI communication

The method used to distribute spikes in groups of MPI processes with collective calls is similar to that used for point-to-point communication, however, an additional structure is used in this case: for each process in a given group, an array containing the indexes of source neurons with outgoing connections to target neurons of other processes in the same group, $H_{\alpha,\sigma,i}$, is created in each process of the same group. Here α is the group index, σ is the rank of the process in which the source neuron is located, $i \in \{0,\ldots,N_{\alpha,\sigma}-1\}$ is the position in the array, $N_{\alpha,\sigma}$ being the array size, and $H_{\alpha,\sigma,i}$ is the index of the source neuron in process σ. This array is sorted in ascending order, i.e. $H_{\alpha,\sigma,i} \geqslant H_{\alpha,\sigma,i-1}$ $\forall i \in \{1,\ldots,N_{\alpha,\sigma}-1\}$, and allows to efficiently keep track of remote source neurons already used in previously created connections. A second array, $I_{\alpha,\tau,\sigma,i}$, aligned with the previous one, represents the local index, in the process of rank τ in the MPI group α, of the image neuron corresponding to the remote source neuron $H_{\alpha,\sigma,i}$, if the latter node has outgoing remote connections directed to target neurons allocated in the process τ, while by convention it is set equal to -1 if the source neuron does not have an image neuron in τ. Furthermore, while for point-to-point communication each source neuron was associated with a sequence of MPI target processes, in this case each node s is associated to the sequence of MPI groups $G_{\sigma,s,j}$ in which it has an image and, for each group, to the corresponding position $i \leftarrow Q_{\sigma,s,j}$ of the node in the sequence $H_{\alpha,\sigma,i}$.

$G_{\sigma,s,j}$ are the groups to which the source neuron must report that it has produced a spike when it emits one. The collective communication function used in this work is `MPI_Allgather`. With this function, each process in an MPI group communicates the spikes of its source neurons simultaneously to all other processes in the same group, while listening for those sent by others. Figure 2 illustrates how the spikes are communicated within MPI groups through collective communication.

4.1.3. Creation of the remote connection structures for point-to-point MPI communication

Remote connections can be created dynamically by a method of the simulator class, `RemoteConnect`$(\sigma,\mathbf{s},\tau,\mathbf{t},p,\mathcal{C},\mathcal{D},\alpha)$ where

- σ is the source MPI process rank;
- $\mathbf{s} = (s_0,\ldots,s_{N_{\text{source}}-1})$ is the array of the indexes of the source neurons, $N_{\textbf{source}}$ is the number of source neurons used in the call;
- τ is the target MPI process rank;
- $\mathbf{t} = (t_0,\ldots,t_{N_{\text{target}}-1})$ is the array of the indexes of the target neurons, $N_{\textbf{target}}$ is the number of target neurons used in the call;
- p is the receptor port index;
- $\mathcal{C}$ is the connection rule dictionary [44];
- $\mathcal{D}$ is the synaptic parameters dictionary;
- α is the MPI group; a special value ($\alpha = -1$ in our work) for this argument is used to create remote connections for point-to-point MPI communication.

Special cases arise when $\mathbf{s}$ and/or $\mathbf{t}$ are sequences of consecutive integers, i.e. $s_i = s_0 + i$ and/or $t_i = t_0 + i$. In particular, the case where $\mathbf{s}$ is such a sequence allows the sequences $\mathbf{S}_{\tau,\sigma}$, $\mathbf{R}_{\tau,\sigma}$ and $\mathbf{L}_{\tau,\sigma}$ to be updated more quickly, thanks to the fact that these sequences are sorted according to the index of the source neuron (equations (1) and (3)).

The `RemoteConnect` method launches the CUDA kernels [45] that build the structures that are used to extract the spikes that must be communicated remotely from the GPU memory of the source process and create the connections [30] and the structures that are used to deliver the spikes in the GPU memory of the target MPI process. As we will see in section 4.1.6, in order to save GPU memory some of these structures can optionally be stored in CPU memory, at the expense of increasing network creation time and dynamics simulation time. To account for this possibility, GPUDirect communication was not implemented in this work, although on some hardware it could significantly reduce communication time between GPUs. Interprocess communication was based on the MPI C/C++ libraries.

To represent the local indexes of image neurons corresponding to the remote neurons $\mathbf{s}$, the `RemoteConnect` method initially allocates GPU memory to hold a temporary array, $\mathbf{l} = (l_0,\ldots,l_{N_{\text{source}}-1})$ with the same size as $\mathbf{s}$.

Many connection rules prescribe extraction of the source and/or target neuron index of each connection from the sequences $\mathbf{s}$ and $\mathbf{t}$ in a nondeterministic manner through pseudo-random number

generators (e.g. *random, fixed in-degree* and *random, fixed total number*) [44]. In some cases, the source neurons that are used in the newly-created connections are only a fraction of the set specified by the `RemoteConnect` call. In order to save GPU memory, it can be convenient to flag the nodes that are actually used, i.e. nodes that have at least one outgoing connection in the target MPI process; only these nodes should be inserted in the maps that associate remote source neurons to local images. This step is not necessary for connection rules that always use all source neurons (as *all-to-all*, *one-to-one*, *random, fixed out-degree*). Furthermore, the increase in network construction time associated with this step may not be justified if the fraction of source neurons that are not used is small. The decision on whether to use this step is a compromise between GPU memory usage and building time; for connection rules in which the source neuron indexes are extracted randomly (e.g. *random, fixed in-degree* and *random, fixed total number*) this decision can be based on the heuristic approach of using a threshold ξ on the ratio between the estimated number of newly-created connections and the size of the set of source neurons specified by the `RemoteConnect` call (e.g. $K_{\text{in}} \times N_{\textbf{target}}/N_{\textbf{source}} < \xi$ for the *random, fixed in-degree* connection rule). In this work we used a threshold $\xi = 1$, unless differently specified. In case the flagging approach is used, on both the source and target MPI processes, a temporary array of boolean values is created in GPU memory, b_i, with $i \in \{0, \ldots, N_{\textbf{source}} - 1\}$ and with all elements initially set to `false`.

The connections outgoing from the image neurons are created in the target MPI process using a `Connect` method, in similar way as for local connections [30], however instead of using the actual set of source neuron indexes specified by the `RemoteConnect` call, the integers from 0 to $N_{\textbf{source}} - 1$ are temporarily used. This is advantageous because in a later step it will be necessary to set the source neuron of each new connection to the index of the corresponding local image neuron, and as we will see later this index can be quickly obtained by knowing its position in the $\mathbf{s}$ sequence. Furthermore, in case the source neuron indexes have to be extracted using a probabilistic rule, the aligned random number generator array $\texttt{RNG}_{\sigma,\tau}$ is used for this purpose. As mentioned previously, the source MPI process uses a variant of the `RemoteConnect` method, which performs only the extraction of the source neuron indexes. For each new connection, we set $b_i = \text{true}$, with the index $i \in \{0, \ldots, N_{\textbf{source}} - 1\}$ that identifies the source neuron of the connection in the sequence $\mathbf{s}$. In this way, only the source neurons that are actually used by the new connections are labeled as `true`. An array $\tilde{\mathbf{u}}$ is then created containing the indexes of the source neurons s_i for which $b_i = \texttt{true}$: $\tilde{\mathbf{u}} = (i : b_i = \texttt{true})$, and a subarray, $\tilde{\mathbf{s}}$, is then extracted from $\mathbf{s}$ using these indexes:

$$\tilde{s}_j := s_{\tilde{u}_j} \tag{4}$$

with $j \in \{0, \ldots, \widetilde{N}_{\textbf{source}}\}$, where $\widetilde{N}_{\textbf{source}}$ is the number of source neurons actually used in the new connections. $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{s}}$ are sorted in ascending order with respect to the values of the elements of $\tilde{\mathbf{s}}$.

Each element of the array $\tilde{\mathbf{s}}$ is searched in $R_{\tau,\sigma,i}$ to see if the source neuron it represents is already mapped to an image neuron in the target MPI process. In this case, the index of the image neuron obtained from the map is assigned to the corresponding element of the array $\mathbf{l}$. In formulas:

$$\text{If } \exists i \text{ such that } R_{\tau,\sigma,i} = \tilde{s}_j, \quad \text{then set } l_{\tilde{u}_j} := L_{\tau,\sigma,i}. \tag{5}$$

Otherwise, if no entry for $\tilde{s}_j$ exists in the map, a new (virtual) image neuron is created in the target MPI process τ. This is done by appending the remote source neuron index $\tilde{s}_j$ to the sequence $\mathbf{R}_{\tau,\sigma}$, appending the current number of nodes M_τ to the sequence $\mathbf{L}_{\tau,\sigma}$, and then increasing M_τ by one:

$$\begin{aligned} \mathbf{R}_{\tau,\sigma} &\leftarrow \left(\mathbf{R}_{\tau,\sigma}, \tilde{s}_j\right), \\ \mathbf{L}_{\tau,\sigma} &\leftarrow \left(\mathbf{L}_{\tau,\sigma}, M_\tau\right), \\ l_{\tilde{u}_j} &:= M_\tau, \\ M_\tau &\leftarrow M_\tau + 1. \end{aligned} \tag{6}$$

After completing this procedure, the map $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ is again sorted in ascending order with respect to the values of $\mathbf{R}_{\tau,\sigma}$ (equation (3)).

Finally, the source neuron indexes of all new connections, for which the positions in the $\mathbf{s}$ sequence were temporarily used, are replaced with the corresponding image neuron indexes, l_i.

In the source MPI process, the source process variant of the `RemoteConnect` method only generates the sequence of source node indexes of the new connections, using the pseudo-random number generator $\texttt{RNG}_{\sigma,\tau}$, aligned with the target process, as discussed previously. Those indexes are used to build the array $\tilde{\mathbf{s}}$ with the same procedure described for the target MPI process. The elements of this array are

used to update the sequence $\mathbf{S}_{\tau,\sigma}$: if no entry for $\tilde{s}_j$ exists in this sequence, then this value is appended to it:

$$\mathbf{S}_{\tau,\sigma} \leftarrow \left(\mathbf{S}_{\tau,\sigma}, \tilde{s}_j\right). \tag{7}$$

After this procedure is completed, $\mathbf{S}_{\tau,\sigma}$ is sorted in ascending order. This ensures that $\mathbf{S}_{\tau,\sigma}$ and $\mathbf{R}_{\tau,\sigma}$ are always aligned (equation (1)).

In the simulation preparation phase, the sequences $\mathbf{S}_{\tau,\sigma}$ are used to extract, for each source neuron index s, a sequence $\mathbf{T}_{\sigma,s}$ of the target MPI processes where s has an image neuron, and to associate to each element of this sequence, $T_{\sigma,s,j}$, the corresponding position $P_{\sigma,s,j}$ of the source neuron index in the sequence $S_{\tau,\sigma,i}$. More specifically, for each target process rank, τ, and for each index i in the range $[0, N_{\tau,\sigma} - 1]$, we call s the index of the source neuron represented by the i-th element of the sequence $\mathbf{S}_{\tau,\sigma}$

$$s \leftarrow S_{\tau,\sigma,i} \tag{8}$$

and we append the target process rank, τ, and the index i to the sequences $\mathbf{T}_{\sigma,s}$ and $\mathbf{P}_{\sigma,s}$, respectively:

$$\begin{aligned} \mathbf{T}_{\sigma,s} &\leftarrow (\mathbf{T}_{\sigma,s}, \tau) \\ \mathbf{P}_{\sigma,s} &\leftarrow (\mathbf{P}_{\sigma,s}, i). \end{aligned} \tag{9}$$

The position i of the source neuron will be the same in the map $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ in the target MPI process, because these sequences are aligned (equation (1)). As discussed previously, to deliver a spike from the source neuron s to the target process $\tau \leftarrow T_{\sigma,s,j}$, its corresponding positions $i \leftarrow P_{\sigma,s,j}$ is sent to this process, where it is used to extract the corresponding image neuron index $L_{\tau,\sigma,i}$, and thus to deliver the spike through the connections outgoing from this node.

4.1.4. Creation of the remote connection structures for collective MPI communication

Maps that associate remote source neurons to their local images in the target MPI process are also created for collective MPI spike communications, except that in this case an additional index, α, is needed to represent the MPI group:

$$(R_{\alpha+1,\tau,\sigma,i}, L_{\alpha+1,\tau,\sigma,i}) \tag{10}$$

is the map that associates the indexes of the source neurons of the MPI process of rank σ in the α MPI group to the corresponding indexes of the image neurons in the target MPI process of rank τ in the same group. As we wrote previously, by convention, the value $\alpha = -1$ is used to indicate point-to-point communication. Thanks to this convention, the definitions of R and L given for point-to-point communications can be merged with those used for collective communications by setting

$$\begin{aligned} R_{0,\tau,\sigma,i} &:= R_{\tau,\sigma,i} \\ L_{0,\tau,\sigma,i} &:= L_{\tau,\sigma,i}. \end{aligned} \tag{11}$$

The procedure used to construct these maps and to create image neurons and remote connections is the same as described for the point-to-point case.

However, for collective communication, it is not necessary to create the sequence $\mathbf{S}_{\tau,\sigma}$ in the source MPI process.

Instead, the set $\mathcal{H}_{\alpha,\sigma}$ of the source neurons of the process of rank σ in the group α passed as arguments to the `RemoteConnect` calls is created in all MPI processes of this group: for each invocation of the `RemoteConnect` method, $\mathcal{H}_{\alpha,\sigma}$ is updated according to the rule:

$$\mathcal{H}_{\alpha,\sigma} \leftarrow \mathcal{H}_{\alpha,\sigma} \cup \{s_0, \ldots, s_{N_{\text{source}}-1}\}. \tag{12}$$

In the simulation preparation phase, the elements of $\mathcal{H}_{\alpha,\sigma}$ are placed in a sequence, $\mathbf{H}_{\alpha,\sigma}$, sorted in ascending order.

$$\mathbf{H}_{\alpha,\sigma} = \mathbf{sort}(\mathcal{H}_{\alpha,\sigma}). \tag{13}$$

In each process τ in the MPI group α, an array, $\mathbf{I}_{\alpha,\sigma,\tau}$, is then created with the same dimensions as $\mathbf{H}_{\alpha,\sigma}$. This array is intended to contain the local image neuron indexes corresponding to the source neurons inserted in $\mathcal{H}_{\alpha,\sigma}$ that have an image in the process τ. The elements of $\mathbf{I}_{\alpha,\sigma,\tau}$ are initialized to

-1; this value remains unchanged for nodes that do not have an image. The $(R_{\alpha+1,\tau,\sigma,i}, L_{\alpha+1,\tau,\sigma,i})$ map is then used to assign the values of $\mathbf{I}_{\alpha,\sigma,\tau}$:

$$\text{If } \exists i \text{ such that } R_{\alpha+1,\tau,\sigma,i} = H_{\alpha,\sigma,j}, \quad \text{then set } I_{\alpha,\sigma,\tau,j} := L_{\alpha+1,\tau,\sigma,i}. \tag{14}$$

The pairs $(H_{\alpha,\sigma,i}, I_{\alpha,\sigma,\tau,i})$ can be used as a map that associates the remote source neuron index to its local image neuron index.

In the source MPI process σ, the sequences $\mathbf{H}_{\alpha,\sigma}$ are used to extract, for each source neuron index s, a sequence $\mathbf{G}_{\sigma,s}$ of the MPI groups where s has an image neuron, and to associate to each element of this sequence, $G_{\sigma,s,j}$, the corresponding position $Q_{\sigma,s,j}$ of the source neuron index in the sequence $H_{\alpha,\sigma,i}$, through the following procedure: for each MPI group, α, and for each index i in the range $[0, \mathbf{size}(\mathbf{H}_{\alpha,\sigma}) - 1]$, we call s the index of the source neuron represented by the i-th element of the sequence $\mathbf{H}_{\alpha,\sigma}$

$$s \leftarrow H_{\alpha,\sigma,i} \tag{15}$$

and we append the MPI group index α and the index i to the sequences $\mathbf{G}_{\sigma,s}$ and $\mathbf{Q}_{\sigma,s}$, respectively:

$$\begin{aligned} \mathbf{G}_{\sigma,s} &\leftarrow (\mathbf{G}_{\sigma,s}, \alpha) \\ \mathbf{Q}_{\sigma,s} &\leftarrow (\mathbf{Q}_{\sigma,s}, i)\,. \end{aligned} \tag{16}$$

To deliver a spike from the source neuron s to the MPI group $\alpha \leftarrow G_{\sigma,s,j}$, its corresponding positions $i \leftarrow Q_{\sigma,s,j}$ is sent by a collective call to the MPI group α. This position is used by each target MPI process of the same group to extract the local image neuron index, $I_{\alpha,\sigma,\tau,i}$, and thus to deliver the spike through the connections outgoing from this node.

4.1.5. Random, fixed in-degree connection rule for neuron populations distributed across multiple MPI processes
The `RemoteConnect` method described previously requires specifying the MPI process σ where the source neurons are located and the MPI process τ where the target neurons are located. However, in many cases, it is useful to represent populations of neurons distributed across multiple MPI processes, and apply probabilistic connection rules to these populations as a whole. This is the case of the scalable balanced network model used in this work and described later. To this end, we can define a distributed population of source neurons, $\mathcal{S}$, as a collection of K subpopulations $\mathbf{s}^{(0)}, \ldots, \mathbf{s}^{(K-1)}$ allocated in K MPI processes, of rank $\sigma_0, \ldots, \sigma_{K-1}$

$$\mathcal{S} = \left\{ \left(\sigma_0, \mathbf{s}^{(0)}\right), \ldots, \left(\sigma_{K-1}, \mathbf{s}^{(K-1)}\right) \right\} \tag{17}$$

and similarly a distributed population of target neurons, $\mathcal{T}$, as a collection of H subpopulations $\mathbf{t}^{(0)}, \ldots, \mathbf{t}^{(H-1)}$ allocated in H MPI processes, of rank $\tau_0, \ldots, \tau_{H-1}$

$$\mathcal{T} = \left\{ \left(\tau_0, \mathbf{t}^{(0)}\right), \ldots, \left(\tau_{H-1}, \mathbf{t}^{(H-1)}\right) \right\}. \tag{18}$$

The connection rule *random, fixed in-degree with multapses* [44] with $K_{\text{in}} = c$ prescribes that for each neuron in the target population, c connections are created by randomly extracting the source neuron of each of these connections with a uniform distribution from all neurons in the source population. Since in the approach used in this work, connections are created in the MPI process where the target neuron is located, this operation can be performed independently for each process τ_j, randomly extracting, for each target neuron of this process, $t_i^{(j)}$, c elements of the set $\mathcal{S}$. This procedure produces $c \times N_{\textbf{target}}^{(j)}$ triplets, where $N_{\textbf{target}}^{(j)}$ is the number of neurons in the jth target subpopulation:

$$\mathcal{C} = ((\widetilde{\sigma}_0, \widetilde{s}_0, t_0^{(j)}), \ldots, (\widetilde{\sigma}_{cN_{\text{target}}^{(j)}-1}, \widetilde{s}_{cN_{\text{target}}^{(j)}-1}, t_{N_{\text{target}}^{(j)}-1}^{(j)}). \tag{19}$$

These triplets are sorted using as a first sort key $\widetilde{\sigma}_i$ and as a second $\widetilde{s}_i$.

$$\mathbf{C} = \mathbf{sort}\,(\mathcal{C}) \tag{20}$$

and the resulting sequence, $\mathbf{C}$ is divided into distinct subsequences based on the value of the source process, σ. For each of these subsequences, the `RemoteConnect` method described previously is then invoked, using as an argument the σ value associated with the subsequence to specify the source MPI process, and using a special connection rule, *assigned-nodes*, which, instead of generating the indexes of the source and target neurons of the connections, uses already assigned values, in this case the values of s and t of the elements of the subsequence.

4.1.6. GPU memory levels

Four GMLs have been implemented to balance the GPU memory occupation and the time-to-solution. Indeed, when performing large-scale simulations of networks with natural neuron and connection density, a significant part of the GPU memory is occupied by data structures for mapping remote connectivity, and some of them may be stored in CPU memory to aim for GPU memory optimization. These GMLs can be employed both with point-to-point and collective communication approaches. The levels, ordered by increasing GPU memory usage, are designed as follows:

- **GML 0**: The maps of the remote source neurons, the maps to their local-images, the first index, and the number of the outgoing connections of each remote neuron are stored in CPU memory. Note that the connections are sorted based on the index of the source neuron as the first sorting key [30], consequently to identify all the connections outgoing from a given neuron, it is sufficient to indicate the index of the first of these connections and their number.
- **GML 1**: The maps of the remote source neurons to their local images, the first index, and the number of the outgoing connections of each remote neuron are stored in CPU memory. From this level on, it is assumed that all source nodes specified as arguments to a `RemoteConnect` call should have an image in the target MPI process, without checking if they are actually used in at least one connection in such a process. This makes the remote connection creation faster. However, if a significant fraction of the source nodes is not actually used in any connection, this leads to a waste of GPU memory. For some connection rules, such as *one-to-one*, *all-to-all* and *fixed-in-degree*, it is guaranteed that each source neuron of the set passed as argument of a call to the Connect method is actually used in at least one connection. On the other hand, there are rules, such as *fixed-in-degree* and *fixed-total-number*, for which it is not necessarily true that every source neuron given as an argument to the call to the Connect method is actually used in some connections.

  Creating an image neuron for each source neuron of the set given as argument of the `RemoteConnect` command without checking whether or not a connection is used induces a slight waste of GPU memory, which in the case of a random network is relatively small when the number of MPI processes is smaller than the average number of outgoing connections for each neuron, but starts to become relevant when the number of MPI processes is comparable or larger than this number.
- **GML 2**: The maps of the remote source neurons to their local images and the first index of the outgoing connections of each remote neuron are stored in GPU memory. The number of outgoing connections of a remote neuron is computed on the fly when needed.
- **GML 3**: The maps of the remote source neurons to their local images, the first index, and the number of the outgoing connections of each remote neuron are stored in GPU memory.

The GML 2 is used as the default for simulation in NEST GPU.

4.2. Models employed for performance evaluation

4.2.1. Multi-area model

The MAM is a large-scale spiking model representing 32 areas of the macaque vision-related cortex [22, 32], each one representing a cortex volume obtained by extruding 1 mm^2 patch through the full cortex thickness. The areas are chosen among those that have visual function or have strong connections with the latter, and are designed with a laminar structure, with layers 2/3, 4, 5, and 6 containing an excitatory and an inhibitory population each, with an intra-area connectivity similar to that of the cortical microcircuit of [46]. The only exception is area TH, which lacks layer 4. The inter-area connectivity is based on axonal tracing and quantitative tracing data, and the neuron densities per area is determined from empirical measurements, cytoarchitectural type definitions of areas, and the thicknesses of the cortical layers. The firing rates of the populations are adjusted through a mean-field approach [47], and the initial membrane potentials of the neurons are normally distributed as in [22].

Based on the synaptic strength of the cortico-cortical synapses, the model can fall into different activity states. When the synaptic strengths are equal between local and cortico-cortical synapses, the model shows a stationary *ground state* of activity, characterized by low activity fluctuations and limited interactions between neurons of different areas. As the cortico-cortical synaptic strength increases, the model shows irregular activity with higher rates and enhanced inter-area interactions, named *metastable state*. Further details on the model may be found in the original publications [22, 32]. Such a model is simulated using the point-to-point communication approach. To eliminate the effects of transient

dynamics at the beginning of the simulation and to assess the performance in more stable conditions, each simulation was preceded by a warm-up of 500 ms of model time, followed by a simulation model time of 10 000 ms. The simulation time resolution was 0.1 ms. The MAM simulations used the default GML (i.e. level 2) to balance between memory occupation and time-to-solution.

Area packing In our initial study with NEST GPU [15], the MAM was implemented so that each area is contained within a single GPU. This leveraged the fact that intra-area connectivity is higher than inter-area connectivity, hence the biggest spike communication payload of each area remained contained within each GPU device and lowered the total bandwidth required for each MPI communication round. At the time, the GPUs used for the study were the NVIDIA V100s (see section 4.5), which also limited the number of simultaneously instantiated areas within a single GPU. With larger GPUs available, such as the NVIDIA A100s (see section 4.5), it stands within reason to explore different area distribution schemes to study whether the increase in neuron network density per GPU and consequently the decrease of MPI communication would enable better performance. As a first network distribution scheme, we implemented an '*area packing*' algorithm to distribute multiple areas within a single GPU while balancing the load across each GPU. The algorithm is based on the classic '0–1 Knapsack problem' [48] where each area can only be assigned once and the area weight is measured by the sum of the total number of incoming connections to the area and the number of neurons within the area. As the implementation relies on knowing the connectivity information of the model before instantiating neurons and connections, the *area packing* algorithm is run during the initialization of the model script (through the Python interface) and scans the connectivity data files included in the model implementation.

4.2.2. Scalable balanced network.

The scalable balance network was closely modeled after the 'HPC benchmark' from NEST CPU, historically used as a performance reference for computing clusters. Unlike the original CPU-based implementation (see [20, 49]), the GPU version of the network model instantiates static excitatory-to-excitatory connections. The average firing rate of excitatory and inhibitory populations results below 10 Hz and does not depend on the network size. In line with the CPU-based implementation, the GPU version also employs collective MPI communication, since densely connected large-scale simulations strongly benefit from using this protocol for the exchange of spikes among the MPI processes. A warm-up time of 500 ms was simulated to discard transients, with the following simulation time being of 1000 ms. The simulation time resolution was 0.1 ms.

Details of scalable architecture In the model implementation in NEST GPU, the number of incoming connections per neuron is fixed to $K_{\text{in}} = 11\,250$, with $K_{\text{in},E} = 9000$ excitatory and $K_{\text{in},I} = 2500$ inhibitory connections, whereas the number of neurons on each MPI process depends on a scaling parameter so that the model has $11\,250 * \texttt{scale}$ neurons, with $9000 * \texttt{scale}$ excitatory and $2250 * \texttt{scale}$ inhibitory neurons.

Thus, the total network size is not only determined by the `scale` parameter, but also by the number of processes used. As opposed to the CPU implementation, where the `scale` parameter determines the total network size, regardless of the number of processes used in the computing system.

4.3. Simulation phases and optimization

The time-to-solution of the simulation can be divided into different phases: *network construction* and *simulation* (or *state propagation*). The former comprises all the steps needed before the simulation loop begins, and it can be further divided as follows:

- Initialization, i.e. the time needed for preparing the model and the simulator, importing the libraries and the classes needed to perform the simulation;
- Neuron and device creation, which instantiates neurons and devices, such as spike generators;
- Neuron and device connection. This can be divided into
  - local connection, which handles the connection among neurons and devices belonging to the same MPI process;
  - remote connections, which handles the connection among neurons simulated in different MPI processes;

- Simulation preparation, needed to organize the connections and initialize data structures for spike delivery.

The time taken by network construction represents an overhead that is less relevant for simulations with relatively long biological simulated times, but can become a significant bottleneck for shorter model times, particularly when a large number of simulations need to be run, e.g. for model parameter optimization.

After network construction is completed, the simulation loop can start. The time needed to complete the simulation loop is measured through the (RTF), defined as the ratio between the wall clock time (T_{wall}) and the biological time simulated (T_{model}):

$$\text{RTF} = \frac{T_{\text{wall}}}{T_{\text{model}}}. \tag{21}$$

When the simulations are performed for benchmarking purposes, the recording of spikes or any variables from neurons and synapses is disabled. Moreover, times are measured at the Python interface level to be agnostic to the implementation.

Another key concept for the optimization of SNN simulations on GPUs is memory usage. GPUs necessarily need to interact with the CPU, so in general, simulations will use both CPU and GPU memory. However, in GPU-based approaches, CPU memory is typically underutilized, partly because most resources are allocated to the GPU and CPU memory is typically much larger even in GPU-equipped cluster nodes. Therefore, the limiting factor in these systems is GPU memory. Typically, in models of neuroscience interest, the number of connections is several orders of magnitude greater than the number of neurons (in the cerebral cortex, typically on the order of $10^3 - 10^4$, while in the cerebellum, even 10^5), so the memory required for connection parameters is much greater than that required for neurons. Appendix D assesses the scaling performance across biologically plausible in-degree ranges. This analysis demonstrates that our approach is ultimately limited by the total number of synapses fitting into memory and not by the number of synapses per neuron. In addition, a significant amount of memory may be required to store the previously mentioned structures used to efficiently communicate and distribute spikes between neurons located in different MPI processes. When simulating large networks, it is essential to monitor how the memory used by these structures scales with the size of the network.

4.4. Validation of spiking statistics

The novel approach for network construction on multi-GPU systems leads to a change in the random numbers in the probabilistic connection rules. Even if the same seed is used, the resulting network instance is different than in the previous version of the simulation code, and spiking activity can be compared only on a statistical level. Thus, it is needed to verify that the network dynamics of the models simulated are qualitatively preserved.

To validate the new network construction method, we performed simulations of the MAM, comparing the results of the simulations obtained using the two versions of our reference implementation. Indeed, *offboard* version has already been validated in [15] in the simulation of the MAM taking the CPU version of NEST as a reference. In this work, we employ the same spike statistics validation protocol, extracting three statistical distributions of the spiking activity for each population of the model:

- Time-averaged firing rate for each neuron;
- Coefficient of variation of inter-spike intervals (CV ISI);
- Pairwise Pearson correlation of the spike trains obtained from a subset of 200 neurons for each population.

The distributions obtained using the different network construction methods are then compared as described in detail in appendix A.

4.5. Hardware

We performed the simulations on two data centers, with different GPU configurations. In particular, we used JUSUF [33] compute cluster from the Jülich Supercomputing Centre, and the Leonardo Booster [3]

from the CINECA supercomputing Centre. The following shows the specifications of these two systems on a per-node basis:

- JUSUF: 2 × AMD EPYC 7742, 2× 64 cores, 2.25 GHz; NVIDIA V100, 1530 MHz, 16 GB HBM2e, 5120 CUDA cores
- Leonardo Booster: 1 × Intel Xeon Platinum 8358 CPU, 32 cores, 2.6 GHz (Icelake); 4 × NVIDIA custom Ampere A100 GPU, 64 GB HBM2, 6912 CUDA cores

JUSUF uses CUDA version 12.0, whereas Leonardo Booster uses CUDA version 12.1. Moreover, the NVIDIA A100 GPUs of the Leonardo Booster show a GPU memory that is four times higher than that of the JUSUF GPU cards. This is relevant especially for strong scaling experiments, since each GPU can handle the simulation of a larger network.

Acknowledgments

The authors gratefully acknowledge the use of Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the Grant Agreement No. 800858. We acknowledge the CINECA award under the ISCRA initiative, for the availability of high-performance computing resources and support. The authors are also grateful to Daniele di Bari for his support in the usage of the CINECA computing infrastructure.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://doi.org/10.5281/zenodo.17627864 [57].

Funding

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement No. 945 539 (Human Brain Project SGA3), the European Union's Horizon Europe Programme under the Specific Grant Agreement No. 10 114 7319 (EBRAINS 2.0 Project), the Italian Ministry of University and Research (MUR) PNRR project FAIR PE0000013- CUP I53C22001400006, funded by NextGenerationEU, MUR PNRR project e.INS Ecosystem of Innovation for Next Generation Sardinia—spoke 10 - CUP F53C22000430001 - MUR code: ECS00000038, the BRAINSTAIN project, funded by INFN-CSN5, and the Juelich Research Centre intramural STEF fund for the update of instruments.

Appendix A. Validation details

As described in section 4.4, the new network construction method needs a validation protocol to ensure that models created using probabilistic connections and having the same parameters show the same network dynamics from a statistical point of view. In order to achieve this, we collect the spiking activity of every neuron population of the multi-area model of [32], computing for each of them three distributions: the average firing rate of the populations, the coefficient of variation of inter-spike intervals (CV ISI), and the pairwise Pearson correlation of the spike trains for each population.

Every simulation has a pre-simulation phase of 500 ms followed by a simulation of 60 s of network dynamics. The pre-simulation phase is needed to avoid spiking activity transients that naturally occur at the simulation's beginning. In contrast, the simulation phase simulates 60 s of neural activity in order to let the activity statistic converge. Indeed, [50] shows that relatively long network dynamics is needed to distinguish the actual activity statistics of a network from a random process.

Only the spikes emitted during the latter phase are recorded and employed to obtain the aforementioned distributions. Figure 7 depicts the violin plots of the distributions obtained for one of the 32 areas of the model in the metastable state. As shown in [15], such a state of the network dynamics is characterized by a high degree of variability even when performing multiple simulations on the same simulation code using different seeds for random number generation.

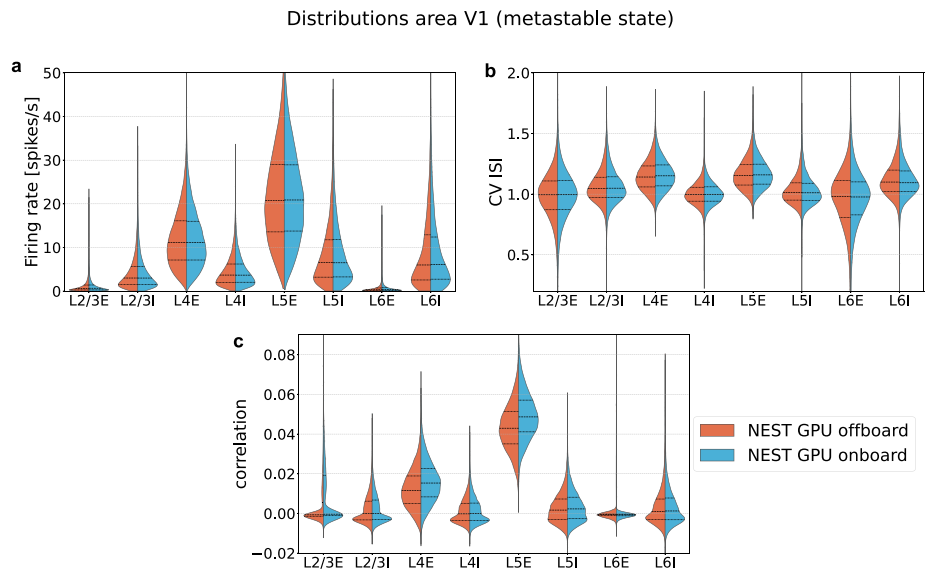

Figure 7. Violin plots of the distributions of firing rate (a), CV ISI (b), and Pearson correlation (c) for a simulation for the populations of the cortical microcircuit model with (sky blue distributions, right) or without (orange distributions, left) the new method for network construction. The distributions are obtained with the `seaborn.violinplot` function of the Seaborn [51] Python library, which smooths the distributions using the Kernel Density Estimation method [52, 53] with Gaussian kernel and with bandwidth optimized through the Silverman method [54].

The distributions obtained by the two versions of the library are mostly similar, however, some differences may occur. The differences that arise between two distributions can be quantitatively estimated using the Earth mover distance (EMD) metric, which is equivalent to the first-order Wasserstein distance and can be computed using the `scipy.stats.wasserstein_distance` of the SciPy Python library [55]. We then followed the validation protocol described in [15], which aims to evaluate the degree of fluctuations that arise between the statistical distribution when simulations are performed through the same simulation code using different seeds and the ones between simulations performed using the two different simulation codes. If the latter fluctuation is compatible with the former, it indicates that the novel methods implemented in the simulation code do not add variability to the statistical results.

The degree of the fluctuation is measured through EMD for every statistical distribution and every population of the model. For a better comparison, three sets of simulations have to be performed using different seeds for random number generation: two sets of simulations with the *offboard* version, and a set of simulations with the *onboard* version, that implements the novel construction method. With this, we can compare the simulations performed with the *offboard* version to evaluate the fluctuations that arise from the change of the seed, whereas one of the sets of simulations just mentioned is compared with the set of *onboard* simulations to estimate the fluctuation between the simulation codes with different network construction methods. The comparisons are made in a pairwise fashion, hence, for each population and statistical distribution, we collect an amount of values of EMD equal to the number of simulations performed for each set.

In this case, every set of simulations is composed of 20 simulations. This number is the result of a compromise between the need for more data to evaluate the fluctuations measured with the EMD and the amount of compute time and storage needed to perform the simulations required. For instance, three sets of 20 simulations of one minute of spiking activity recorded in the metastable state of the MAM need up to 20 TB of memory. Moreover, the compute time needed, especially for the simulations with the previous method for network construction, is an additional limiting factor.

Figure 8 shows the values of EMD for the three statistical distributions for the first area of the MAM.

The box plots of figure 8 show a similar distribution of the values of EMD for the comparisons performed, which leads to the conclusion that the novel method for network construction does not add variability to the statistical distribution with respect to the previous version of the library.

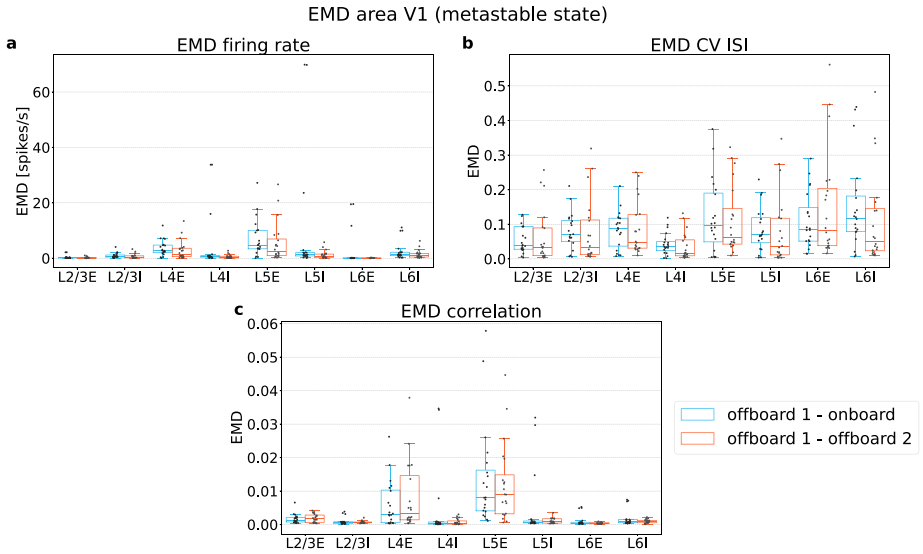

Figure 8. Box plots of the Earth Mover's Distance comparing side by side firing rate (a), CV ISI (b) and Pearson correlation (c) of the two versions of the code (sky blue boxes, left) and the previous version using different seeds (orange boxes, right). The central line of the box plot represents the median of the distribution, whereas the extension of the boxes is determined by the interquartile range of the distribution formed by the values of EMD of each comparison. The whiskers show the rest of the distribution as a function of the interquartile range, and the dots represent the 20 values of the EMD from which the boxes are generated.

Appendix B. MAM area packing results

The area packing algorithm described in section 4.2.1 was developed to fit more areas of the MAM model inside the same GPU to enable GPU memory-optimized simulations of the MAM. To this aim, the NVIDIA A100 GPUs of the Leonardo Booster supercomputer [3], with 64 GB of GPU memory, can be used to test the performance of the MAM when this algorithm is employed. Indeed, the V100 GPUs of the JUSUF cluster, with 16 GB of GPU memory, could fit up to a model area per GPU. Figure 9 shows the performance benchmarks of the MAM when area packing is employed.

The algorithm enables the MAM to be executed on two compute nodes with 4 GPUs each (i.e. 2 nodes), even if with a longer time-to-solution with respect to the implementations in which the model areas are simulated into a larger number of GPUs. Figure 9(b) depicts the real-time factor (RTF), which aligns to the one shown in figure 3(b) when the same number of GPUs is employed. Indeed, despite different GPUs are used in the two cases, the RTF of the simulation can be similar with these devices, as also shown in [30] for the simulation of the model used as a building block for the MAM. Figure 9(c) shows that the increased network construction time for few MPI processes is mainly due to the increased load on the area packing algorithm. The plateau occurs at eight nodes, thus in a condition in which 32 GPUs are used. Indeed, this is expected since the difference between the last three configurations is only the number of GPUs used on each compute node.

Although NVLink 3.0 (used for intra-node GPU communications [3]) offers higher bandwidth than InfiniBand, no major performance gain was observed, as inter-node communication remains the limiting factor. An improvement in this regard could be achieved by simulating the entire model in a single compute node, which is not possible with the memory limits of the GPUs used here. Besides, one could explore more specialized communication schemes such as the NVIDIA Collective Communications Library for GPUs if suitable high-speed interconnects within a node or NVIDIA networking across nodes is available (https://developer.nvidia.com/nccl, last visited: 24.10.2025).

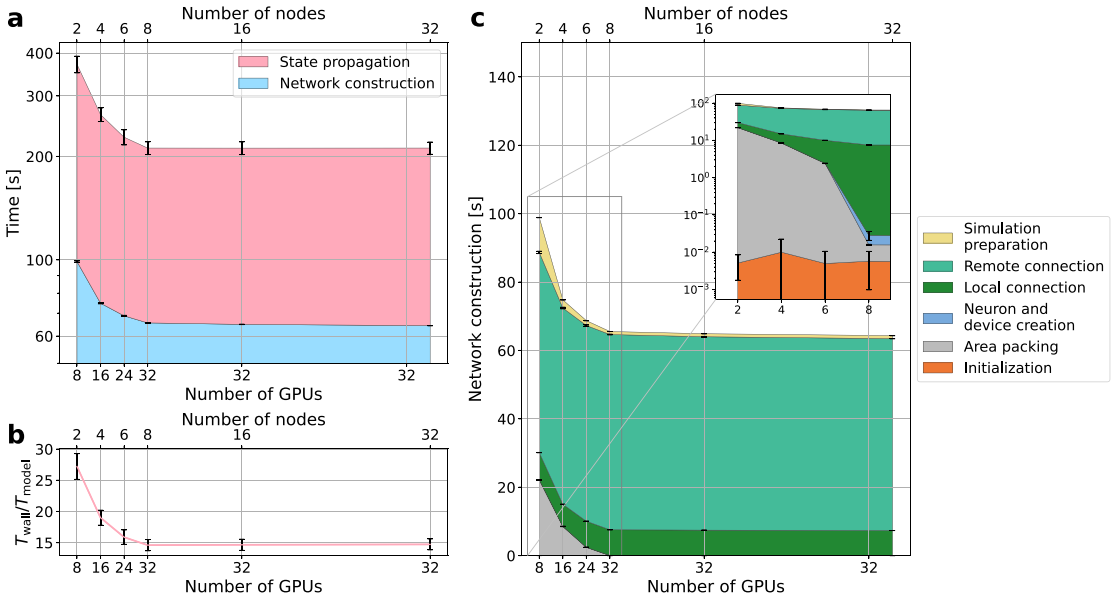

Figure 9. Performance of the *onboard* version on simulation of MAM metastable state leveraging the area packing algorithm on Leonardo Booster. Mean data shown from averaging over 10 simulations using different random seeds, black error bars represent the standard deviations. For the 2–8 node configurations, four GPUs per node were utilized. For the 16-node configurations, two GPUs per node were used, and for the 32-node configuration, one GPU per node was used. All configurations use 1 MPI process per GPU. (a) Absolute wall-clock time for the network construction and state propagation for a biological model time $T_{\mathrm{model}} = 10$ s. (b) State propagation described as real-time factor. (c) Network construction time divided into subtasks.

Appendix C. Scalable balanced network simulations using different network scale parameter

In the main body of the manuscript, we performed simulations of the scalable balanced network model setting the scale parameter to 20, i.e. 2.25×10^5 neurons and 2.53×10^9 synapses per GPU. However, different scales can be adopted to simulate larger or smaller networks inside each node. To provide more insight into the network construction performance of networks of different sizes, figures 10 and 11 show construction and simulation preparation time in case of networks of scale 10 (i.e. 1.125×10^5 neurons and 1.26×10^9 synapses per GPU) and 30 (i.e. 3.375×10^5 neurons and 3.80×10^9 synapses per GPU), respectively. As can be seen, the network size, expressed in terms of total synapses, differs from that of figure 6.

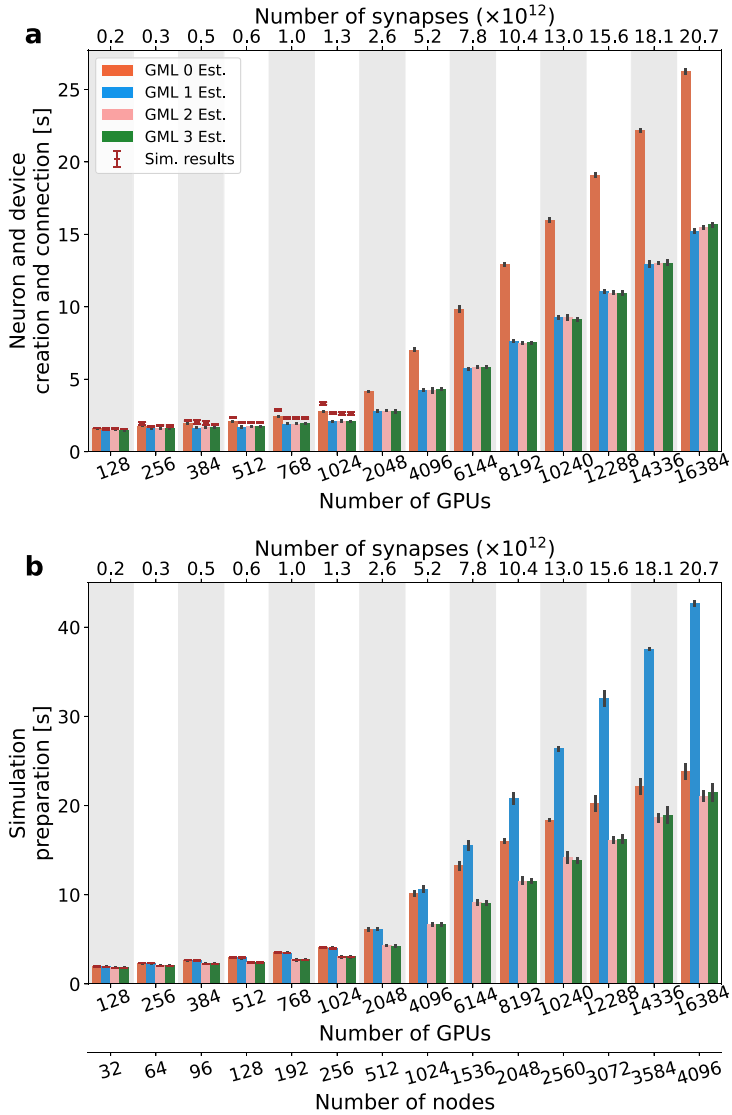

Figure 10. Network construction time divided into neuron creation and connection (a) and simulation preparation (b) times of the scalable balanced network model simulated at scale 10 evaluated across all MPI processes as a function of the number of cluster nodes for the three GPU memory levels. The bars represent an estimation using four MPI processes on a single node. The horizontal line markers show the time averaged across the MPI processes using the GPU memory levels. Errors for bars and horizontal line markers represent their standard deviation, with most of the values showing little variability.

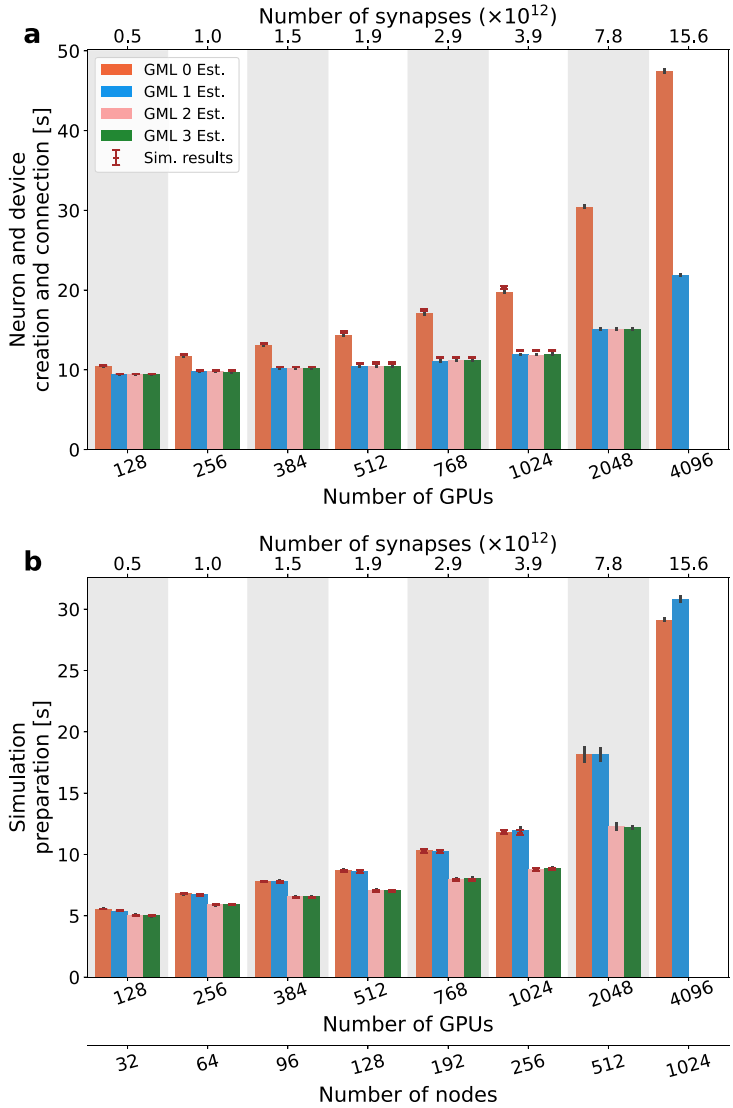

Figure 11. Network construction time divided into neuron creation and connection (a) and simulation preparation (b) times of the scalable balanced network model simulated at scale 30 evaluated across all MPI processes as a function of the number of cluster nodes for the three GPU memory levels. The bars represent an estimation using four MPI processes on a single node. The horizontal line markers show the time averaged across the MPI processes using the GPU memory levels. Errors for bars and horizontal line markers represent their standard deviation, with most of the values showing little variability.

Appendix D. Scalable balanced network with different in-degree scale parameter

The scalable balanced network described in section 4.2.2 shows a fixed number of incoming connections per neuron, with the `scale` parameters used to change the number of neurons simulated on each MPI process. This way, the network size increases, both in the total number of neurons and connections. However, a different approach may be employed to test the capabilities of the simulation code by simulating a network when the number of incoming connections per neuron changes. Neurons in certain regions of the brain, such as the cerebellum [56], may show up to 2×10^5 incoming connections, and similar in-degree levels are not explored in the simulation of the network models proposed in this work, and it would be of interest to show the performance of the simulation code in such a framework.

In this regard, we performed simulations on a modified version of the scalable balanced network model. We defined a `in-degree_scale` parameter, so that each MPI process has a total of $9000 \times$ `scale/in-degree_scale` excitatory neurons and $2250 \times$ `scale/in-degree_scale` inhibitory neurons, with `scale` $= 10$ and a fixed in-degree per neuron equal to $K_{\text{in}} = 11\,250 \times$ `in-degree_scale`. Moreover, to preserve the average firing rate of the model, the synaptic weights for each connection are adjusted with the `in-degree_scale` factor, so that the product of the number of in-degrees and their synaptic weights remains constant. This way, the total number of connections per MPI process is the

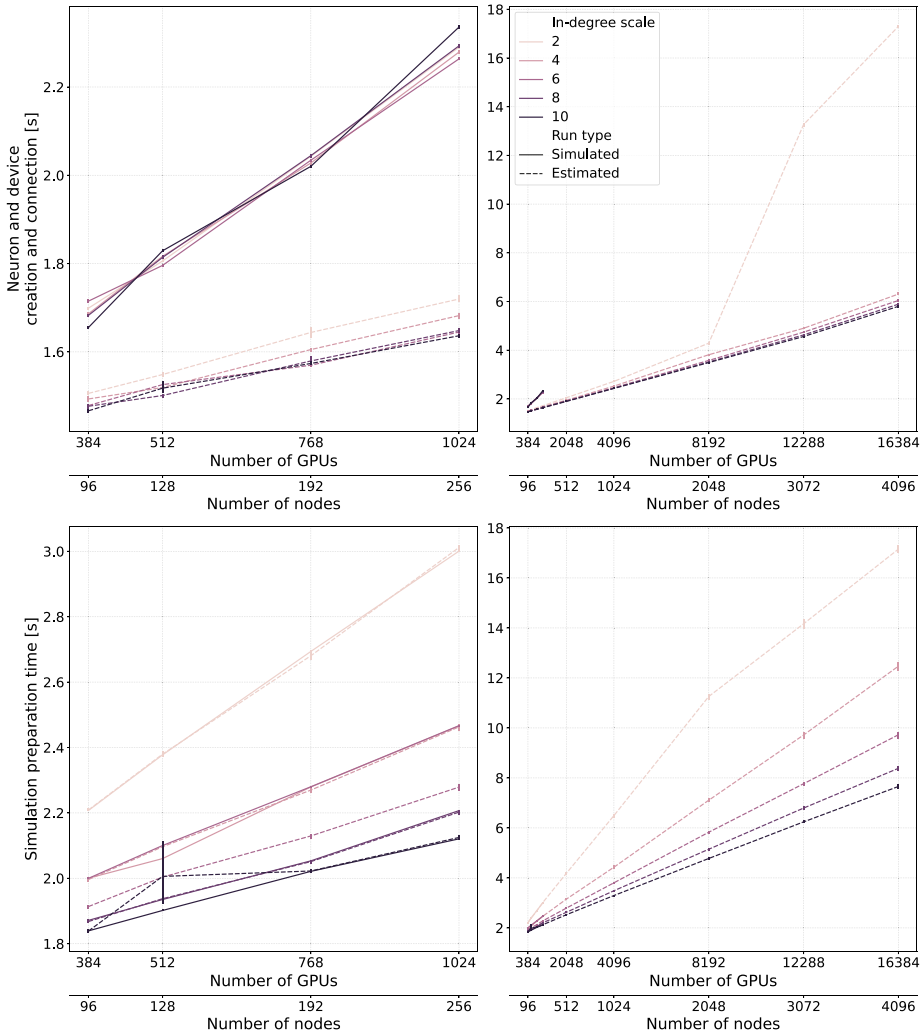

Figure 12. Neuron and device connection and creation and simulation preparation times of the scalable balanced network model for different values of `in-degree_scale` as a function of the number of cluster nodes (four GPUs per node). Panels on the left represent the results in the range of compute nodes for which simulations are executed, whereas right panels extend to the range in which the performance is evaluated with the estimation method. Lines represent simulated results, whereas dotted lines represent estimated results. Vertical bars indicate the standard deviation.

same as for the scaling experiments described in the Results, granting similar GPU memory consumption. Similarly, the firing rate does not show significant differences with respect to the scalable network model configurations employed in the rest of the work.

This experiment has been performed using the GML 0 since, as shown in the Results, it is the level capable of leveraging the largest number of compute nodes. Figure 12 shows neuron and device creation and simulation preparation times using different values for `in-degree_scale`. The maximum value adopted for such a scaling parameter is 10, i.e. the in-degree values range from 22 500 for `in-degree_scale` = 2 to 112 500 for `in-degree_scale` = 10.

The figure shows that both times diminish as soon as the `in-degree_scale` increases. This is expected, since a larger parameter implies that the number of neurons diminishes, and thus also the number of image nodes, resulting in faster construction and sorting operations.

Appendix E. Analysis of the difference between simulated and estimated values for the scalable balanced network model

In this appendix, we provide a detailed analysis of the discrepancies observed between estimated and simulated values of GPU memory consumption and during neuron and device creation and connection times (figures 5 and 6).

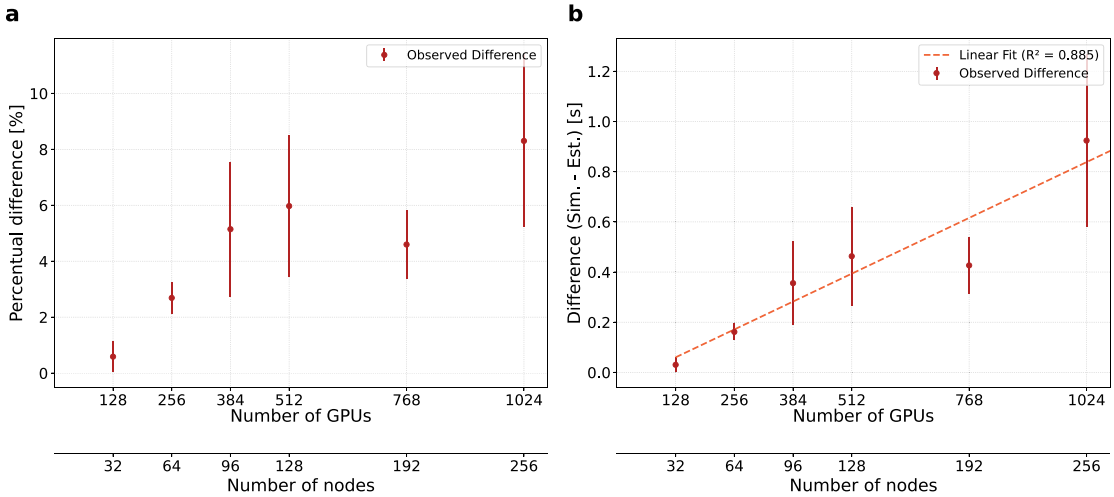

Figure 13. Difference between simulated and estimated neuron and device creation and connection time for GPU memory level 0, measured as percentual difference (a) or absolute difference (b). Dots represent the observed values, whereas the dotted line in panel (b) represents a linear fit of such differences. This data is derived from the simulations at the scalable network models using `scale` = 20, thus using the data of figure 6.

Regarding the divergence observed in figure 5 for the GPU memory peak, the construction of the network and of the maps for remote connectivity involves the use of temporary buffers in GPU memory, which are allocated in a non-deterministic manner due to the randomness of the connectivity operations. Therefore, the GPU memory usage peak shows slight variations (in the order of 1%) across different MPI processes, which is mostly observable in the case of GMLs 2 and 3, where connection structures are explicitly stored and indexed in GPU memory.

This also affects the performance of the network construction; however, there are other sources of variability in the latter case, especially at the GML 0. Large-scale distributed systems are subject to performance jitter, i.e. fluctuations in execution time caused by operating system activity and hardware-level variability. These effects accumulate with the number of nodes and are not captured by the estimation model. Additionally, during the simulations, the CPU thread responsible for connection construction is not bound to a specific core. Consequently, operating system scheduling may migrate the thread across cores, inducing cache invalidation, thus increasing variability in execution time.

Differences between simulated and estimated network construction times in figure 6 are shown in figure 13. Figure 13(a) shows the percentage difference between simulated and estimated time, showing values for 256 nodes below 10%. Figure 13(b) shows the same discrepancy in absolute terms, and a linear fit has been performed to assess the degree of the discrepancies when thousands of compute nodes are employed for the simulation. Extrapolations of such behavior indicate discrepancies of about 14 s for 4096 nodes.

Appendix F. Example of remote spike communication through point-to-point MPI forwarding

Figure 14(a) illustrates how the maps $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ are organized in the GPU memory of an MPI process, which has rank $\tau = 1$ in this example. This process creates one map for each possible source MPI process, i.e. one for every MPI rank excluding its own (ranks $\sigma = 0$ and $\sigma = 2$ in this example). These maps are organized in fixed-size blocks that are allocated dynamically in GPU memory.

Figure 14(b) illustrates how the sequences $S_{\sigma,\tau,i}$ are organized in the GPU memory of a different MPI process, which has rank $\sigma = 0$ in this example. One sequence of this kind is created for each possible target MPI process, i.e. one for every MPI rank excluding that of the source process itself (ranks $\tau = 1$ and $\tau = 2$ in this example). Note that we chose to show two different MPI processes in the two panels to better illustrate the correspondence between the $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ maps, allocated in the target MPI process, τ, and the $S_{\sigma,\tau,i}$ sequences, allocated in the source MPI process, σ. Indeed, it can be

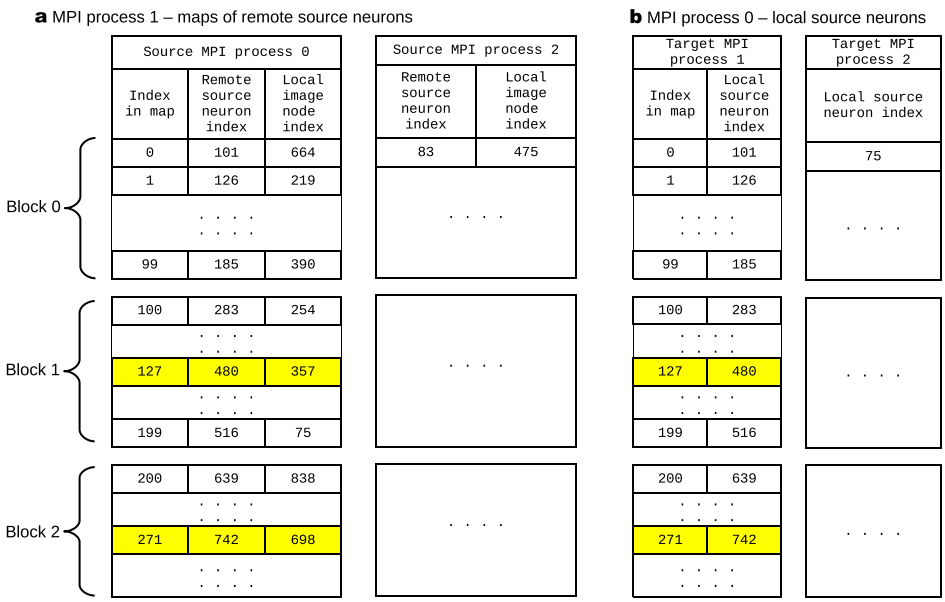

Figure 14. (a) organization of the maps $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$, which associate remote source neurons to local image neuron indexes, in the GPU memory of the target MPI process, which has rank $\tau = 1$ in this example. The left column represents the position in the map, i, while the other two columns represent the remote source neuron index, $R_{\tau,\sigma,i}$ and the local index of its image neuron in the target MPI process, $L_{\tau,\sigma,i}$. (b) organization of the sequence $S_{\sigma,\tau,i}$, which represent the local source neuron index, in the GPU memory of the source MPI process, which has rank $\sigma = 0$ in this example. The left column represents the position in the sequence, i, while the other column represent the local source neuron index, $S_{\sigma,\tau,i}$.

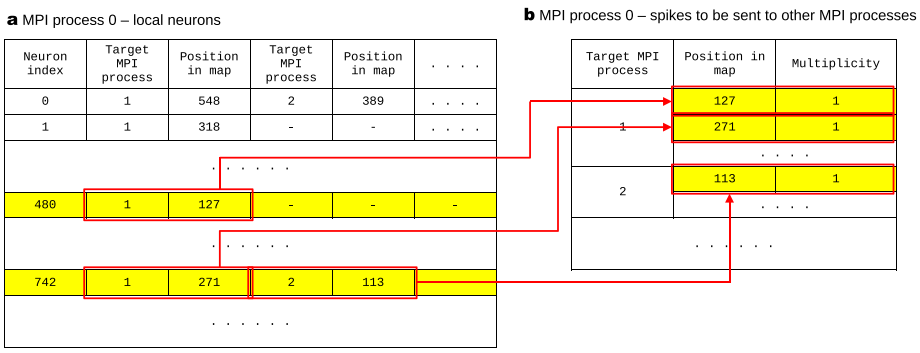

Figure 15. (a) organization of the sequences $T_{\sigma,s,j}$, which represent the ranks of the target MPI processes in which each node has an image, and $P_{\sigma,s,j}$ which are the corresponding positions of the neuron in the maps that associate it to its image neurons in the target processes. (b) creation of the packets to be sent to target MPI processes using point-to-point MPI messages. In this example we assume that in a certain time step nodes 480 and 742 of the MPI process 0 emit a spike. We can observe that node 480 has an image in the target process 1, and that the corresponding position in the maps is 127, while node 742 has images in the MPI processes 1 and 2, and the corresponding positions are 271 and 113, respectively. These positions are appended to the packets created for the respective target MPI processes, which will be sent via point-to-point communication.

observed that the remote source neuron indexes in the map for the source process 0 allocated in the target process 1 are equal to the local source neuron indexes of the sequence $S_{\sigma,\tau,i}$ allocated in the source process 0 for the target process 1, in agreement with equation (1). In this example, we assume that at a certain time step the neurons of index 480 and 742 of the MPI process of rank 0, which are highlighted in yellow in the figure, emit a spike.

Figure 15(a) shows how the sequences $T_{\sigma,s,j}$ and $P_{\sigma,s,j}$ are organized in the GPU memory. The first column contains the source neuron index, s, while the other columns represent alternatively the values of the different target MPI processes $T_{\sigma,s,j}$ in which the node s has an image, and the positions $i \leftarrow P_{\sigma,s,j}$ of the neuron in the maps $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ that associate the source neuron to its image neurons in the target processes.

Figure 15(b) illustrates how the spikes are inserted in packets that are sent to the target processes through point-to-point MPI communications.

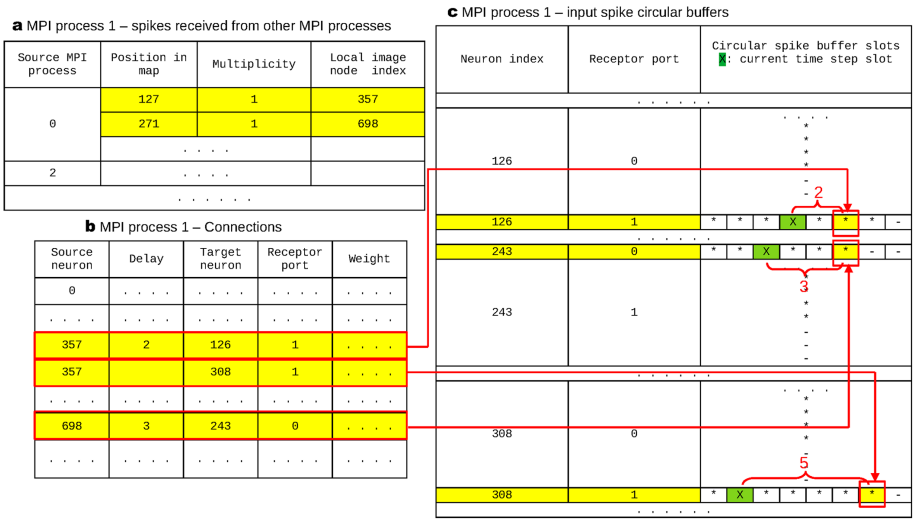

Figure 16. Delivery of the spikes received by the target MPI processes. (a) The first column indicates the source MPI process, the second column represents the position i (127 and 271 in this example) of the remote source neurons that emitted the spike in the map $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$ that associate the source neuron to its image neurons in the target processes, the third column is the spike multiplicity, and the fourth column contains the values of $L_{\tau,\sigma,i}$, which represents the local image neuron indexes. (b) Connection structures allocated in the MPI process 1, sorted according to the source neuron index [30]. The connections outgoing from local images of neurons that emitted a spike are highlighted in yellow. In this example, the node 357 has two outgoing connections, directed to the target neurons 126 and 308, with delays 2 and 5 in time-step units, respectively, while the node 698 has one outgoing connection, directed to the target neuron 243, having delay 3 in time-step units. (c) The spikes are inserted into the input circular spike buffers of the target neurons, in a slot shifted from the one representing the current time step by a number of positions equal to the delay (2, 3 and 5 in the example, written in red). The spikes are accumulated in these slots by adding, for each of them, its multiplicity multiplied by the connection weight.

Finally, figure 16 shows how the spikes received by the target MPI process are delivered to the target neurons using the maps $(R_{\tau,\sigma,i}, L_{\tau,\sigma,i})$, through the connection structures and the input spike circular buffers of the target neurons.

ORCID iDs

Bruno Golosio 0000-0001-5144-6932
Gianmarco Tiddia 0000-0001-7524-0285
José Villamar 0009-0007-8791-7100
Luca Pontisso 0000-0001-7137-5254
Luca Sergi 0009-0006-4633-186X
Francesco Simula 0000-0002-7955-1491
Pooja Babu 0000-0002-1963-6370
Elena Pastorelli 0000-0003-0682-1232
Abigail Morrison 0000-0001-6933-797X
Markus Diesmann 0000-0002-2308-5727
Alessandro Lonardo 0000-0002-5909-6508
Pier Stanislao Paolucci 0000-0003-1937-6086
Johanna Senk 0000-0002-6304-062X

References

[1] Senk J *et al* 2026 Constructive community race: full-density spiking neural network model drives neuromorphic computing *Neuromorph. Comput. Eng.* **6** 012001
[2] Aimone J B, Awile O, Diesmann M, Knight J C, Nowotny T and Schürmann F 2023 Editorial: neuroscience, computing, performance and benchmarks: why it matters to neuroscience how fast we can compute *Front. Neuroinf.* **17** 1157418
[3] Turisini M, Cestari M and Amati G 2024 Leonardo: a pan-European pre-exascale supercomputer for HPC and AI applications *J. Large-Scale Res. Facil.* **9** A186
[4] Message Passing Interface Forum 2009 MPI: a message-passing interface standard, version 2.2 *Technical Report* (available at: http://www.mpi-forum.org/docs)
[5] Lu W *et al* 2024 Simulation and assimilation of the digital human brain *Nat. Comput. Sci.* **4** 890–8

[6] Du X, Wang M, Lu Z, Duan Q, Liu Y, Feng J and Wang H 2024 HRCM: a hierarchical regularizing mechanism for sparse and imbalanced communication in whole human brain simulations *IEEE Trans. Parallel Distrib. Syst.* **35** 1056–73
[7] Yavuz E, Turner J and Nowotny T 2016 GeNN: a code generation framework for accelerated brain simulations *Sci. Rep.* **6** 18854
[8] Knight J C and Nowotny T 2021 Larger GPU-accelerated brain simulations with procedural connectivity *Nat. Comput. Sci.* **1** 136–42
[9] Beyeler M, Carlson K D, Chou T S, Dutt N D and Krichmar J L 2015 Carlsim 3: a user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks *Proc. Int. Joint Conf. on Neural Networks (IJCNN)* pp 1–8
[10] Niedermeier L, Chen K, Xing J, Das A, Kopsick J, Scott E, Sutton N, Weber K, Dutt N and Krichmar J L 2022 Carlsim 6: an open source library for large-scale, biologically detailed spiking neural network simulation *2022 Int. Joint Conf. on Neural Networks (IJCNN)* pp 1–10
[11] Carnevale N T and Hines M L 2006 *The Neuron Book* (Cambridge University Press)
[12] Kumbhar P, Hines M, Fouriaux J, Ovcharenko A, King J, Delalondre F and Schürmann F 2019 CoreNEURON: an optimized compute engine for the neuron simulator *Front. Neuroinf.* **13** 63
[13] Awile O *et al* 2022 Modernizing the NEURON simulator for sustainability, portability and performance *Front. Neuroinf.* **16** 884046
[14] Pronold J, Jordan J, Wylie B J N, Kitayama I, Diesmann M and Kunkel S 2022 Routing brain traffic through the Von Neumann bottleneck: parallel sorting and refactoring *Front. Neuroinf.* **15** 785068
[15] Tiddia G, Golosio B, Albers J, Senk J, Simula F, Pronold J, Fanti V, Pastorelli E, Paolucci P S and van Albada S J 2022 Fast simulation of a multi-area spiking network model of macaque cortex on an MPI-GPU cluster *Front. Neuroinf.* **16** 883333
[16] Wang F 2015 Simulation tool for asynchronous cortical streams (STACS): interfacing with spiking neural networks *Proc. Comput. Sci.* **61** 322–7
[17] Wang F, Kulkarni S, Theilman B, Rothganger F, Schuman C, Lim S-H and Aimone J B 2024 Scaling neural simulations in STACS *Neuromorph. Comput. Eng.* **4** 024002
[18] Gewaltig M-O and Diesmann M 2007 NEST (NEural Simulation Tool) *Scholarpedia* **2** 1430
[19] OpenMP Architecture Review Board 2008 OpenMP application program interface (available at: http://www.openmp.org/mp-documents/spec30.pdf) (Accessed 27 September 2016)
[20] Kunkel S *et al* 2014 Spiking network simulation code for petascale computers *Front. Neuroinf.* **8** 78
[21] Jordan J, Ippen T, Helias M, Kitayama I, Sato M, Igarashi J, Diesmann M and Kunkel S 2018 Extremely scalable spiking neuronal network simulation code: from laptops to exascale computers *Front. Neuroinf.* **12** 2
[22] Schmidt M, Bakker R, Shen K, Bezgin G, Diesmann M and van Albada S J 2018 A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas *PLoS Comput. Biol.* **14** e1006359
[23] Pronold J, van Meegen A, Shimoura R O, Vollenbröker H, Senden M, Hilgetag C C, Bakker R and van Albada S J 2024 Multi-scale spiking network model of human cerebral cortex *Cereb. Cortex* **34** bhae409
[24] Senk J, Hagen E, van Albada S J and Diesmann M 2024 Reconciliation of weak pairwise spike-train correlations and highly coherent local field potentials across space *Cereb. Cortex* **34** bhae405
[25] Gandolfi D, Mapelli J, Solinas S M G, Triebkorn P, D'Angelo E, Jirsa V and Migliore M 2023 Full-scale scaffold model of the human hippocampus CA1 area *Nat. Comput. Sci.* **3** 264–76
[26] Yamaura H, Igarashi J and Yamazaki T 2020 Simulation of a human-scale cerebellar network model on the K computer *Front. Neuroinf.* **14** 16
[27] Kudithipudi D *et al* 2025 Neuromorphic computing at scale *Nature* **637** 801–12
[28] Albers J *et al* 2022 A modular workflow for performance benchmarking of neuronal network simulations *Front. Neuroinf.* **16** 837549
[29] Golosio B, Tiddia G, De Luca C, Pastorelli E, Simula F and Paolucci P S 2021 Fast simulations of highly-connected spiking cortical models using GPUs *Front. Comput. Neurosci.* **15** 627620
[30] Golosio B, Villamar J, Tiddia G, Pastorelli E, Stapmanns J, Fanti V, Paolucci P S, Morrison A and Senk J 2023 Runtime construction of large-scale spiking neuronal network models on GPU devices *Appl. Sci.* **13** 9598
[31] Herten A *et al* 2024 Application-driven exascale: the Jupiter benchmark suite *SC24: Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (IEEE) pp 1–45
[32] Schmidt M, Bakker R, Hilgetag C C, Diesmann M and van Albada S J 2017 Multi-scale account of the network structure of macaque visual cortex *Brain Struct. Funct.* **223** 1409–35
[33] Vieth B V S 2021 JUSUF: modular tier-2 supercomputing and cloud infrastructure at jülich supercomputing centre *J. Large-Scale Res. Facil.* **7** A179
[34] Brunel N 2000 Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons *J. Comput. Neurosci.* **8** 183–208
[35] Reghenzani F, Massari G and Fornaciari W 2020 Timing predictability in high-performance computing with probabilistic real-time *IEEE Access* **8** 208566–82
[36] Wybo W A, Jordan J, Ellenberger B, Mengual U M, Nevian T and Senn W 2021 Data-driven reduction of dendritic morphologies with preserved dendro-somatic responses *eLife* **10** e60936
[37] Boucsein C, Nawrot M, Schnepel P and Aertsen A 2011 Beyond the cortical column: abundance and physiology of horizontal connections imply a strong role for inputs from the surround *Front. Neurosci.* **5** 2011
[38] Rosen B Q and Halgren E 2022 An estimation of the absolute number of axons indicates that human cortical areas are sparsely connected *PLoS Biol.* **20** e3001575
[39] Hagmann P, Cammoun L, Gigandet X, Meuli R, Honey C J, Wedeen V J and Sporns O 2008 Mapping the structural core of human cerebral cortex *PLoS Biol.* **6** e159
[40] Lober M, Diesmann M and Kunkel S 2026 Exploiting network topology in brain-scale simulations of spiking neural networks (arXiv:2602.23274)
[41] Qu P, Lin H, Pang M, Liu X, Zheng W and Zhang Y 2023 ENLARGE: an efficient SNN simulation framework on GPU clusters *IEEE Trans. Parallel Distrib. Syst.* **34** 2529–40
[42] Knight J C, Senk J and Nowotny T 2026 A flexible framework for structural plasticity in GPU-accelerated sparse spiking neural networks *Neuromorph. Comput. Eng.* **6** 014019
[43] Dai K *et al* 2020 The SONATA data format for efficient description of large-scale network models *PLoS Comput. Biol.* **16** e1007696
[44] Senk J, Kriener B, Djurfeldt M, Voges N, Jiang H-J, Schüttler L, Gramelsberger G, Diesmann M, Plesser H E and van Albada S J 2022 Connectivity concepts in neuronal network modeling *PLoS Comput. Biol.* **18** e1010086

[45] NVIDIA Corporation 2024 CUDA toolkit documentation, version 12.5 (available at: https://developer.nvidia.com/cuda-toolkit)
[46] Potjans T C and Diesmann M 2014 The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model *Cereb. Cortex* **24** 785–806
[47] Schuecker J, Schmidt M, van Albada S J, Diesmann M and Helias M 2017 Fundamental activity constraints lead to specific interpretations of the connectome *PLoS Comput. Biol.* **13** e1005179
[48] Martello S and Toth P 1990 *Knapsack Problems: Algorithms and Computer Implementations* (Wiley)
[49] Helias M, Kunkel S, Masumoto G, Igarashi J, Eppler J M, Ishii S, Fukai T, Morrison A and Diesmann M 2012 Supercomputers ready for use as discovery machines for neuroscience *Front. Neuroinf.* **6** 26
[50] Dasbach S, Tetzlaff T, Diesmann M and Senk J 2021 Dynamical characteristics of recurrent neuronal networks are robust against low synaptic weight resolution *Front. Neurosci.* **15** 757790
[51] Waskom M L 2021 Seaborn: statistical data visualization *J. Open Source Softw.* **6** 3021
[52] Rosenblatt M 1956 Remarks on some nonparametric estimates of a density function *Ann. Math. Stat.* **27** 832–7
[53] Parzen E 1962 On estimation of a probability density function and mode *Ann. Math. Stat.* **33** 1065–76
[54] Silverman B W 1986 *Density Estimation for Statistics and Data Analysis* (Chapman and Hall)
[55] Virtanen P *et al* (SciPy 1.0 Contributors) 2020 SciPy 1.0: fundamental algorithms for scientific computing in Python *Nat. Methods* **17** 261–72
[56] Popa L S, Streng M L and Ebner T J 2018 Purkinje cell representations of behavior: diary of a busy neuron *The Neuroscientist* **25** 241–57
[57] Tiddia G *et al* 2026 Scalable Construction of Spiking Neural Networks using up to thousands of GPUs archive (https://doi.org/10.5281/zenodo.17627864)