Norwegian University
of Life Sciences

Master’s Thesis 2022 30 ECTS
Faculty of Science and Technology
Professor Hans Ekkehard Plesser

Performance Analysis of Neuronal Network Connectivity Creation from Complex Rules

AbdulHakeem Ayodele Adamu
MSc. Data Science

Acknowledgments

The writing of this thesis would not have been possible without some amazing people who contributed immensely in ways that I would never be able to put to words.

Firstly, I would like to thank my supervisor Prof. Hans Ekehard Plesser for guiding me every step of the way and patiently listening to all the many issues I faced during the past months writing this thesis. His vast wealth of knowledge and wisdom is the foundation of my work. I am eternally grateful.

I would also like thank Susanne Kunkel, Jochen Martin Eppler, Håkon Mørk, Johanna Senk and Stine Brekke Vennemo for taking time out of their busy schedules to help me out in running my simulations. I extend my gratitude to my colleagues Nida Grønbekk and Sanjayan Rengarajan, whom I could turn to in times of need. A big thank you to the NEST developers and the entire team at JSC.

I 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.

Lastly, I would like to thank my parents and siblings, my dear Tensaye, Cyril, and all my other friends. Without you all I would never have done any of this.

AbdulHakeem Ayodele Adamu
Ås, October 2022

Abstract

This thesis investigates the performance of neuronal network connectivity instantiation during neuronal simulation. It is important to try and get the simulation times of neuronal networks closer to the actual biological time as this will significantly increase the use cases of computational neural network simulations. Since network connectivity creation phase makes up a large fraction of the total time take during network simulation, it is important to try and reduce it (Morrison et al. 2005; Ippen et al. 2017). In this thesis, we perform an investigative analysis of the utilization of compute resources of different neuronal networks during parallel simulations on a supercomputer. This is done in order to identify how key properties of neuronal networks formed from different connection rules contribute to compute-time consumption during network connectivity creation. Due to the connectivity rules, neuronal network models incorporating combinations of different network properties were formed. How these properties affect the consumption of hardware resources is investigated here. We approach this investigation by performing weak and strong scaling experiments involving the simulation of different neuronal networks. These neuronal networks are constructed using NEST, and are performed on a supercomputer using beNNch as a benchmarking tool. The plots gotten from this experimentation are discussed extensively in this paper.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The nervous system is considered the most important biological system as it facilitates the survival of organisms by allowing them adapt to changes in their environments (Society for Neuroscience 2018). Translating light waves, sound waves, vibrations, and food molecules into electrical signals, the nervous system helps organisms perceive the world around them. In mammals, this system is made up of the spinal cord, a network of nerves, and the brain at the heart of it all. The human brain is made up of networks of complex circuits of about 86 billion microscopical cells called neurons (Society for Neuroscience 2018; National Research Council 1989; Smith 1952).

The importance of fully understanding the mechanism of the nervous system cannot be overstated. An estimated one in four people worldwide face a neurological or psychiatric disease yearly (United Nations 2001). Studies and investments have gone into trying to understand the workings of the brain and mental or psychiatric illnesses — whose cost is about 10% of the world's yearly GDP (Markram 2013). Returns from these investments in terms of new treatments and drugs have been slow and seem to be decreasing (Gustavsson et al. 2011; Markram 2013). Furthermore, new qualitative insights building upon invaluable pioneering work from the 1950s into how networks of neurons give rise to observed neural representations have been lacking (Einevoll et al. 2019).

Today, a fairly strong understanding of how individual neurons operate and process information has been established. Neurons interact primarily through action potentials or *spikes*. These spikes are stereotyped point events generated during inter-neuronal communication. They occur after a neuron receives an accumulation of ten to a hundred action potentials within a few milliseconds[1].

[1]The workings of action potentials depend on the type of neurotransmitters produced by the neurons. Statistically, there is a ratio of about 80% to 20% excitatory to inhibitory neuronal population in the human brain. The action potentials of excitatory neurons pushes connecting neurons toward generating spikes, while those of inhibitory neurons suppress connecting neurons from generating spikes. (Sterratt et al. 2011)

The inflow of action potentials eventually causes the electric potential difference between the inside and outside of the neuron to cross the cell-membrane's threshold voltage, thus, releasing an action potential to the next neuron. Action potentials are transmitted repeatedly - several billion times a second across the brain at separation points between neurons called the *synapses* (Sterratt et al. 2011; Society for Neuroscience 2018).

Following work by Hodgkin and Huxley (1952) in understanding of the propagation of action potentials across neurons, biophysics-based modeling of neurons has been established. This provides an important platform which most studies are based upon (Einevoll et al. 2019). These studies have been used in deriving descriptive mathematical models which account for different individual neuronal properties. Due to the multitude and diversity of neurons however, it has been very difficult to effectively theorize how networks of millions or billions of heterogeneous neurons work together at a systemic level to provide brain functions (Einevoll et al. 2019). Nonetheless, mathematical equations based on specific biological details of some areas of the brain, and the types of neurons and synaptic connections present there have brought about the development of several network models which provide insights into various network dynamics (Potjans and Diesmann 2014; Thomson et al. 2002; Binzegger et al. 2004).

Over the years, comprehensive neuronal networks modeling specific parts of the brain have been developed. Small homogeneous neuronal populations with statistically identical connection properties have been modeled as in studies by McCormick and Huguenard (1992, Halnes et al. (2011). Studies using substantially reduced neuronal population sizes of the brain have been conducted as well (Migliore et al. 1995; Brunel 2000). In the widely cited model by Brunel (2000), the brain was simplified into a network of sparsely (randomly) connected neurons with similar statistical properties. They used an 80% to 20% ratio of excitatory and inhibitory neuronal representation. Spatial positioning was disregarded in this model (Brunel 2000). The neuronal modelling approach taken here is useful in understanding basic properties of large networks of neurons (Sterratt et al. 2011).

Although larger networks which consider the heterogeneous nature of neural populations in the brain and their more structured inter-neuronal connections are being modeled; large comprehensive mechanistic network modeling is still in its infancy however (Einevoll et al. 2019). This can be attributed to the sheer size and complexity of mammalian brains. They contain multi-compartmentalized neurons arranged spatially. For even the smallest of mammals, a cubic millimeter of the cortex of their brains contain close to 100,000 neurons. Each of these neurons receive an average of around 10,000 local inputs and another 10,000 inputs from more distant locations (Ippen et al. 2017). Therefore, simulating any spatially-dependent comprehensive network model becomes extremely computationally intensive. Nevertheless, with increasing computational power, and the advent of modern supercomputers, simulations of large comprehensive networks with billions of neurons are becoming expedient.

Our understanding of the mechanism of the brain and its functions have been limited partly due to the sheer amount of cells involved and the intricately detailed connections formed between these cells. Mathematical modeling has been adopted to aid with the qualitative analysis of brain functions. Today, for many cognitive functions, including the most advanced ones, mathematical models have been developed. The challenge lies in scaling up from knowledge of individual mathematical models to gaining a systemic level of understanding. Although, there is still some way to go in gaining qualitative insights about cognitive functions from network models, it remains promising as there have been successful applications of this type of high level scaling in other sciences such as the case of the highly multivariate numerical weather prediction. The accuracy of weather forecasts has increased yearly, and is today, very precise (Bauer et al. 2015).

Computational simulation, an essential component of scientific methods, is used in representing the evolution of a model over time. In neuroscience, it is used to study the relationship between anatomical and physiological data (Einevoll et al. 2019). Neuronal simulation has proven very important in gaining insights from models, and also for testing out different hypotheses. It is required for the complete exploration of the dynamics of almost all the neuronal network models used today (Senk et al. 2021). One of the earliest usage of neuronal simulation was in the pioneering experiment by Hodgkin and Huxley (1952) over 70 years ago, where they propagated an action potential along an axon. Today, with the progress in computer hardware technology, simulations can be used for networks of up to a billion neurons and their corresponding synapses (Kunkel et al. 2014; Jordan et al. 2018).

Due to the large amount of computational resources required by large scale simulations, they are typically run on modern high performance clusters (HPCs) and supercomputers. These computers typically have multiple compute nodes connected by fast interconnects such as infinibands. Each of the compute nodes also contain several multi-core CPUs 1.1. Although these high performance computers have many nodes, poorly written software programs may under utilize the compute resources. Therefore, harnessing the power of modern computational advancement lies in writing optimal simulation code (Abi Akar et al. 2019; Kumbhar et al. 2019). Optimal simulation code should communicate efficiently across all compute nodes so the network model can run in parallel adeptly. In addition, memory also has to be distributed efficiently across all the compute nodes. Every core in a CPU share the same instance of the operating system. This makes parallel simulations within a single compute node possible using threads. For parallelization across multiple compute nodes, communication over a physical network is required.

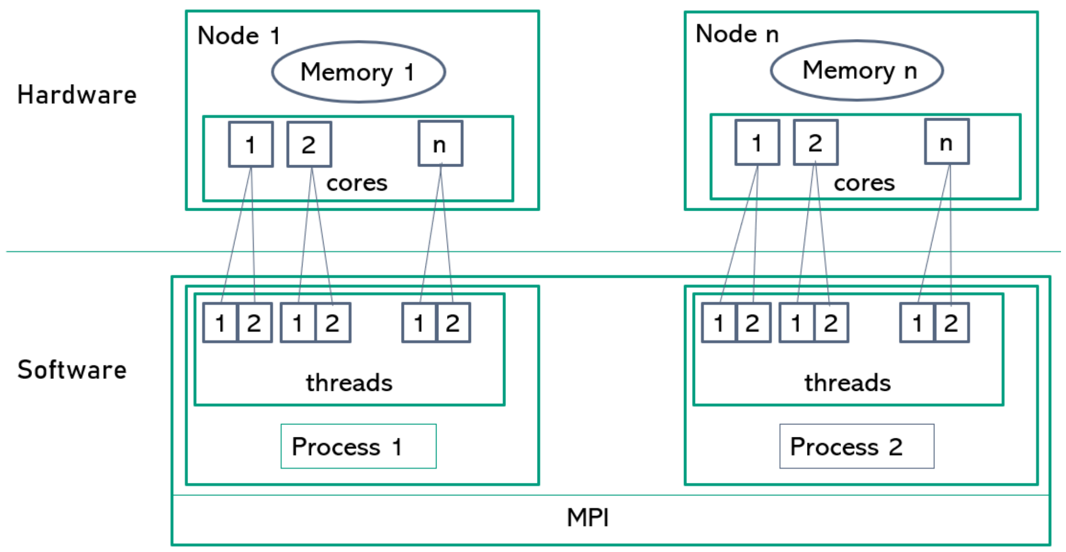

Figure 1.1: Illustration of a typical multiprocessing compute system with MPI interface.

Aims, scope, and organization

As highlighted in Morrison et al. (2005) and Ippen et al. (2017), the fraction of the total time taken during the network connectivity creation phase in large scale simulations is significantly large. Thus, it is important to try and reduce it. *The aim of this thesis* will therefore be to perform an investigative analysis of the utilization of compute resources of different neuronal networks during parallel simulations on a supercomputer. This is done in order to identify how key properties of neuronal networks formed from different connection rules contribute to compute-time consumption during network connectivity creation. Due to the connectivity rules, neuronal network models incorporating combinations of different network properties were formed. The different network properties include: The architecture based on spatial dependence. Here, metrics determining the proximity between neurons could be introduced or not, making it spatial or non-spatial networks respectively; The determinism rule with which connections are either formed or not based on probability functions or explicit rules. How these properties affect the consumption of hardware resources is investigated here.

For the analysis, we perform both weak and strong scaling experiments of the different networks formed from the combination of these connectivity rules. These experiments are done based on the current version of the `NEST` simulator, v3.3 (Spreizer et al. 2022). We get the performance indication from the scalable parallelism. This indicator is gotten from a weak scale experiment done by simulating the neuronal networks on a supercomputer and measuring the time taken to connect the neurons as more resources are added while also simultaneously increasing the network size to scale with the increased allocated

MPI processes.

Section 2.1 introduces the theories and concepts used in this thesis, as well as the key metrics used for computing the performance of our benchmarks in this thesis. The underlying principles network models have, and the different forms of connections the networks may form are also described here. In Chapter 2, we also introduce the simulation software and the various software used in computing the different benchmarks. The specification of the hardware used is also discussed here. Afterwards, the network models we used in our simulation and how we developed them are described. Section 2 closes off with the key steps undertaken in performing the weak scale benchmarking experiments used in this thesis.

Afterwards, we compare the results obtained from our experiments in Chapter 3. These results are the performance indicators from the weak and strong scaling of the simulation of our different neuronal network models. We present these results with the aid of several plots. These results are described objectively here. We finalize this thesis with a discussion of our observations in Chapter 4. Here, we highlight some of our findings and give explanations as to why some of these results were achieved.

Chapter 2

Materials and Methods

2.1 Theory and concepts

2.1.1 Scalability

A large fraction of the total simulation time of a neuronal network model goes into constructing the neuronal network, therefore, all the compute power available should be used (Ippen et al. 2017). Scalability or scaling, is a good indicator of optimal parallelization. It is the attribute of a compute system to handle an increasing amount of work by utilizing additional compute resources to the system. To ensure maximal usage of compute power, it is of great importance to measure the parallel scaling of your code (Li 2018). For simulation code to scale properly (i.e., use up the available resources efficiently), the experimental (actual) *speedup* has to be close to the ideal speedup for a certain amount of compute resources. Speedup is the unit for measurement in parallel computing.

$$speedup = \frac{t_1}{t_N}\ . \tag{2.1}$$

Here t_1 is the computational time for running the code on one processor, and t_N is the computational time for running the same code on all processors. The ideal speedup is achieved when every part of a program is parallelizable and is run on multiple processors. This is a challenging goal for most brain simulators (Li 2018). Efficient parallelization is therefore fundamental for maintaining performance for a given span of biological time (Ippen et al. 2017).

There are two main types of scaling: *Strong scaling* and *weak scaling*. In neuronal network simulations, measuring weak and strong scaling provides a good indication of how to share resources amongst different parts of the simulation software. It also provides information about the usage of computational processes (i.e. threads and MPI processes) available in a supercomputer for a large network model.

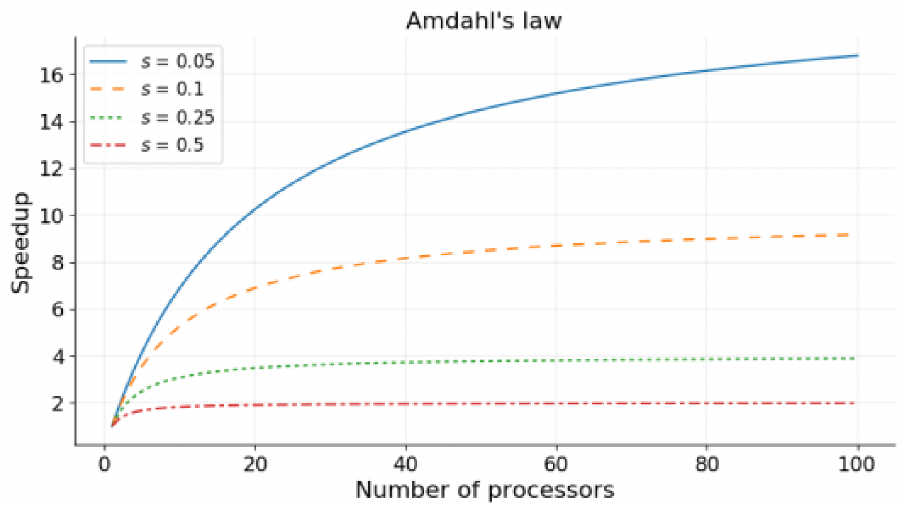

Figure 2.1: Illustration of Amdahl's law. The different lines show the results when you have different fractions of non parallelizable parts of the system.

Strong scaling

Strong scaling is governed by *Amdahl's law*. This law states that "the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used" (Amdahl 1967). This implies that for strong scaling, speedup is limited by the fraction of the code which is not affected by the parallelization;

$$speedup = \frac{1}{(s + \frac{p}{N})} \ . \tag{2.2}$$

Here s is the fraction of the serial (non-parallelizable) part, p is the execution time of the parallelized part, and N is the number of processors. Consider a program which takes 20 hours to run using a single thread. If one part of the program, which takes one hour to run cannot be parallelized ($s = 1/20$), while the rest of the code can ($p = 19/20$), then regardless how many threads are allocated to the program, the minimum execution time cannot be less than one hour. Therefore, in strong scaling, the efficiency of parallelization decreases as the amount of resources increases; making parallel computing with many processors useful for only high parallelized software (Li 2018). Strong scaling is measured by testing how the overall computational time of a job scales with the the number of threads and MPI processes. In our case, this is done by taking the (wall) time in seconds it takes to construct a neuronal network per amount of MPI processes allocated. Figure 1.1 shows the speedup against an increasing number of processors.

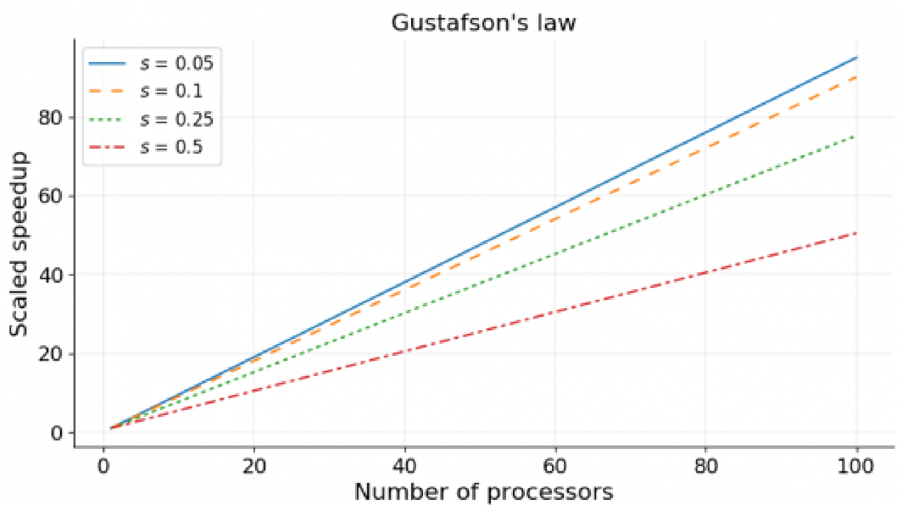

Figure 2.2: Illustration of Gustafson's law. The different lines show the results when you have different fractions of non parallelizable parts of the system.

Weak scaling

Weak scaling on the other hand was defined by Gustafson (1988). This came about as a revision of Amdahl's law which is most well suited to fixed size problems. Weak scaling does not assume that the execution workload does not change with respect to increasing resources. In practice, the problem size scales with the amount of available resources (Li 2018). *Gustafson's law* states that the parallel part of a program scales linearly while the serial part does not increase. In the study, it was stated that increasing the size of problems will allow for the complete exploitation of the computing power brought about by more resources.

$$scaledspeedup = s + p \times N \ . \tag{2.3}$$

Where s, p, N are defined as in strong scaling above. Weak scaling is measured by testing how the overall computational time of a job scales when the size of the job is increased along with the number of threads and MPI processes. In our weak scaling experiments, we increase the network size by multiplying the number of neurons in the network with the number of compute nodes. Thus, affecting the overall number of inter neuronal connections. Figure 2.2 shows the scaled speedup against an increasing number of processors.

2.1.2 Neuronal Network Model

Presently, there is no generally acceptable network model which describes the functioning of the nervous system despite years of studies (Einevoll et al. 2019). It is therefore important to select the network model which satisfies the purpose of the simulation. A neuronal network model can be defined as "an explicit and specific hypothesis about the structure and microscopic dynamics of (a part of) the nervous system" (Nordlie et al. 2009). The complexity of the underlying neuron models - the building blocks of these networks vary different depending on the purpose of the networks.

Neuronal connectivity concepts

Unlike in many other sciences, there is no generally adopted formalism and standardization for representing connectivity in neuronal modeling. There has been some push by the wider neuroscience community to formalize the notations used for describing neuronal connectivity. In this thesis, we shall adopt some of the formalisms proposed by work by Nordlie et al. (2009, Djurfeldt (2012, Senk et al. (2021). In the simulator, a network model is typically modeled as a directed graph. The neurons are represented as nodes while the synapses are represented as directed edges. A population of neurons (multiple nodes) form a structure, and multiple synapses form a projection. The network is connected such that a projection is made between source and target populations using a number of connection rules.

There are typically two types of connection rules[1]. They are deterministic connection rules, and probabilistic connection rules. In deterministic connectivity, connection rules are always precisely defined. Some ways in which connections may be created deterministically include: *One-to-one*, where each source node is connected to exactly one target node; *all-to-all*, where each source node is connected to every target node; and *explicit*, where connections are made based off an explicitly defined map of connections. In probabilistic connectivity, connection rules are defined based on some form of probability function. It may also be precisely defined is in the case of zero probabilistic functions. Some common examples of probabilistic connections are *pairwise Bernoulli*, where connections are made between neurons based on a Bernoulli trial; *fixed total number*, where a fixed number of source-target pairs are chosen at random; *fixed in-degree*, where a fixed number of source nodes are connected to each target node; and *fixed out-degree*, where each source node is connected exactly to a fixed number of randomly chosen target nodes (Senk et al. 2021).

When spatial dependence is included during connectivity, connectivity gets more complex. This complexity arises due to a number of factors which lie in the architectural differences. Non-spatial networks typically have small numbers of neuronal populations with similar neuronal properties connected randomly. They require less memory size than their spatial counterpart and connection creations are less computationally intensive. Spatial networks on the other hand, typically require more compute resources. They represent connections across several areas of the brain and can accommodate topographic

[1]The connection rules may have a number of types based on some properties, in this case however, we separate based on determinism.

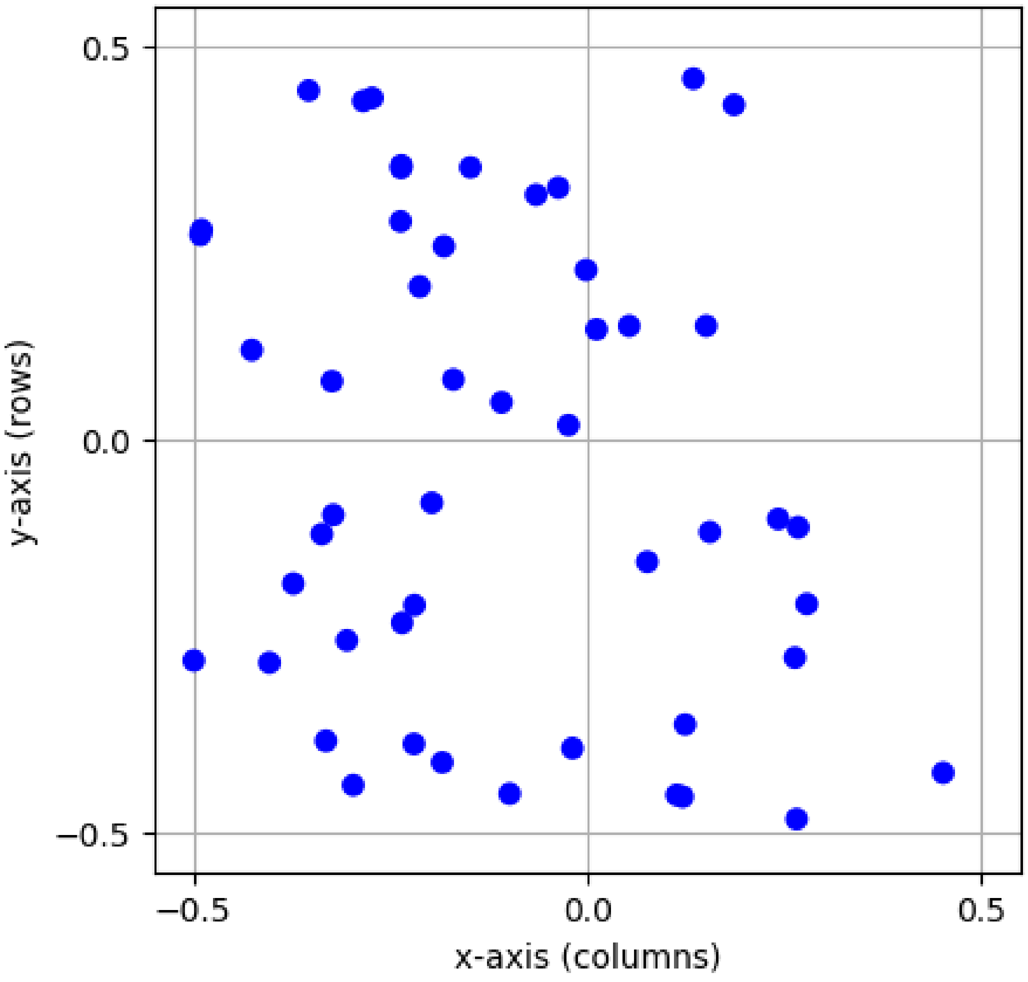

Figure 2.3: A free layer with 50 elements uniformly distributed.

connections (Sterratt et al. 2011). They are more closely analogous to the biological brain than non-spatial networks. Spatial distribution could be constructed in two ways: Grid-based layers and free layers. In grid-based layers, each node is placed at a location in a regular grid with a pre determined distance between them. For free layers, they are placed arbitrarily in the plane. In this project, we used the free layers with random positions in a uniform distribution. The edge wrap is set to true so that the nodes at opposite ends of the plane wrap to form connections between them as if they were laying side by side. This gives a periodic boundary condition. Figure 2.3 shows an illustration of how a free layer with 50 nodes uniformly distributed looks.

In spatial networks, some metric for measuring proximity between nodes is introduced. One very common metric is the radial distance between neurons. This metric determines the layout of the nodes which are included during connection. Spatial dependent connectivity is different from non-spatial connectivity chiefly because of the distinct separation of source and target neural populations during connection[2]. When creating connections an explicit dictionary of connections may also be given so that the connections formed between nodes are predetermined.

With spatial probabilistic connectivity, the calculation of the probability of forming a connection between source and target is the most complex amongst all the connectivity rules discussed before. A statistical distribution is calculated with the positional distance and the boundary condition as parts of the parameters. The probability of forming a connection then reduces the farther away the target neuron is depending on the details of the rule. In our case, we set up this distance metric. Some form of determinism may be mixed in the statistics so that different compartments have different probability values as in the case of (Potjans and Diesmann 2014). A fixed probability may be given for the entire source population so that they do not need to include any distant metric.

Some connectivity constraints may be applied to the different connections or prohibited. Two of the more common constraints are *autapses* and *multapses*. An autapse is when a node can make a connection with itself. A multapse is the situation when a source node A can form a connection to another node B which has connections already to A as the source. Figure 2.4 shows an overview of some of these connectivity concepts discussed. In this thesis we allowed autapses for all of the networks.

2.2 `NEST`, The Neural Simulation Tool

`NEST` (Gewaltig and Diesmann 2007; Spreizer et al. 2022) is a neuronal network simulation software used in optimally simulating large networks of spiking model neurons with relatively simple internal dynamics. There are a large range of different neuronal models and synapses provided by `NEST`. It also provides different high level interfaces which researchers may use to create different neural networks. These interfaces include `NEST`'s own built-in interpreter (SLI), a Python interface (Eppler et al. 2009; Zaytsev and Morrison 2014), and the simulator-independent PyNN interface (Davison et al. 2008). `NEST` also allows users to specify connectivity algebraically using the Connection Set Algebra (CSA) (Djurfeldt 2012). In `NEST`, neuronal networks are represented as directed graphs; neurons and recording devices are represented as nodes; while synapses are represented as edges.

Internally the data structure of the neurons, synapses and networks are implemented in C++. The simulation kernel is also implemented in *C++*. Both the nodes (neurons) and edges (synapses) are instances of their model

[2]This property was noted in a handful of simulators used during the extensive studies conducted in (Senk et al. 2021)

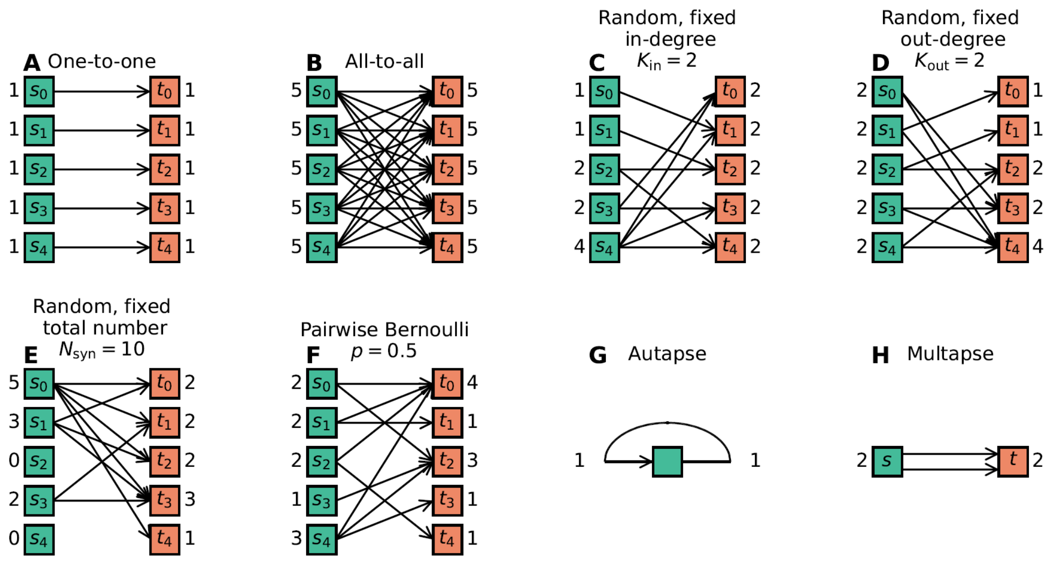
A One-to-one
B All-to-all
C Random, fixed in-degree $K_{in} = 2$
D Random, fixed out-degree $K_{out} = 2$
E Random, fixed total number $N_{syn} = 10$
F Pairwise Bernoulli $p = 0.5$
G Autapse
H Multapse
s_0 s_1 s_2 s_3 s_4
t_0 t_1 t_2 t_3 t_4
s t

Figure 2.4: Overview of common connectivity rules. *From (Senk et al. 2021) with permission.* The green nodes, s_i represent the source nodes while the orange nodes, t_i represent the target nodes. The numbers on the left show the outdegree while the numbers on the tight show the indegree.

classes. The graph (network) is implemented with a strong emphasis on efficient connectivity lookup and memory utilization. This optimal implementation is partly due to the distribution of connectivity storage across multiple compute nodes. A hybrid parallelization scheme (Plesser et al. 2007; Ippen et al. 2017) combines Message Passing Interface (MPI) (Message Passing Interface Forum 2021) for parallelization across multiple compute nodes with Open MP (OpenMP Architecture Review Board 2008) for thread based parallelization. This parallelization scheme is implemented in such a way that it is abstracted away from the researcher. It uses the concept of virtual processes, that is,

$$N_{VP} = M \times T \,. \tag{2.4}$$

Where N_{VP} is the number of virtual process the simulation code is to be distributed over, M is the number of MPI processes, and T is the number of OpenMP threads per MPI processes.

The network nodes are linearly enumerated and distributed amongst the virtual processes using NEST's Global Identifier (GID) in a *round-robin* fashion. The representation of nodes on virtual processes are compact, with a distinct separation in representation of local-nodes from non-local nodes (check (Morrison et al. 2005; Kunkel et al. 2012)). Edges are also represented compactly on the virtual processes using a specialized adaptive data structure minimizing memory overhead (Morrison et al. 2005; Kunkel et al. 2014). Connections are stored on the virtual processes based on the localization of the target nodes, i.e. only target nodes which are local to that virtual process is stored in the virtual process, and so on. Connections are stored using a hierarchical data structure, with source neurons mapped to connections. C++ *STL vectors* and C-style arrays are used in storing connections.

A new connection is registered by checking the sparse table of the VP for the presence of a connector. If the table does not contain a connector, a connection is created and stored, (check Ippen et al. (2017) and Jordan et al. (2018) for a more detailed description of connection creation/deletion and memory allocations/deallocations). After all the connections have been created on the virtual processes, a new neuronal network is built. `NEST` uses `malloc()` to request for memory, and `free()` to return allocated memory. It implements this using *jemalloc*[3]. It has an efficient scheme for handling concurrent requests for memory by different threads by limiting access to the allocator to one thread at a time (Ippen et al. 2017).

The benchmark data used in this thesis are created using revision ed45e47 of `NEST`, version 3.3, available publicly on GitHub (https://github.com/nest/nest-simulator/tree/v3.3), except for benchmark on the tabulated Gaussian connection scheme (Listing 2, Section 2.4) first implemented in pre-release version, revision 7788eee (https://github.com/slimkeem/nest-simulator/tree/v3.3.1).

2.3 Benchmark Scenario

2.3.1 JUSUF Supercomputer

JUSUF (Vieth 2021) is a petaflop supercomputer operated by Jülich Supercomputing Centre (JSC) at Forschungs-zentrum Jülich. The JUSUF system provides HPC system resources as a service demand. It has 205 compute nodes (144 CPU-only nodes and 61 GPU nodes), based on Atos Bull X400 servers. The compute nodes have two AMD EPYC ROME 7742 64-Core processors, 256GB main memory and an 800GB NVMe. The 144 CPU-only nodes have Sequana X440A Double Twin servers while the GPU nodes have Sequana X430A servers equipped with Nvidia Volta V100 GPU per processor. The nodes communicate using dual-port HDR100 Nvidia Mellanox Connect-X6 HCAs InfiniBands. For cloud communication with the storage system (JUSTCOM), Connect-X6 port with 40 Gigabit Ethernet is used (Vieth 2021). More details about the system are provided in Table 2.1.

JUSUF operates on open-source software run on CentOS 7 Linux operating system. The software stack is based on RedHat OpenStack Platform (RedHat OpenStack Platform 2021)). It provides support for ParaStation MPI and OpenMPI. For job management, JUSUF uses the open-source Slurm workload manager (LLC. 2019) in combination with the ParaStation resource management. This offers scalability, reliability and performance for all jobs.

[3]Check (Ippen et al. 2017) to see a comparison of different alternative memory allocators

Table 2.1: JUSUF Configuration

Processor	Two 64-Core AMD EPYC ROME 7742
CPU	256 (16 x 16) GB DDR4, 3200MHz
Node communication	Dual-port HDR100 Nvidia Mellanox (Connect X6) InfiniBand
Main Memory	256GB SSD

We utilize the performance and reliability of the HPC services of JUSUF as the computing hardware for running the neuronal network simulations used in this thesis.

2.3.2 `beNNch`

For large scale neuronal network simulations, benchmarking is typically carried out by assessing the scaling performance of the simulation architecture by increasing the amount of hardware resources while either increasing the size of the network simultaneously as in weak scaling or keeping it fixed as in strong scaling. Some of the dynamics such as the correlation of neuronal networks may change during these scaling experiments (Van Albada et al. 2015). In order to have meaningful benchmarks, it is important that they are comparable at different stages of the simulation. This comparability has proven difficult due to the nature of neuronal network simulations and the difficulty in reproducing the same simulations on other platforms (Senk et al. 2021; Albers et al. 2021).

`beNNch` - a benchmarking framework for neuronal network simulations (Computational and Systems Neuroscience & Theoretical Neuroscience 2021a) is a simulator tool used by neuroscientists for performance benchmarking of neuronal simulations. It works based on a modular workflow consisting of four sequential segments. The first segment is the configuration and preparation segment where all necessary prerequisites are prepared. Here, the configurations such as the amount of compute nodes or number of threads are setup. The second stage is the benchmarking stage where the actual benchmarking occurs. Here, compute intensive calculations are submitted as jobs via scripts holding simulation information. It should be noted that the focus of this benchmarking is the performance result not explicitly the simulation output. In the third and final segments, data is gathered, verified and presented in an intuitive way (Albers et al. 2021). In this thesis, we use `beNNch` to run our neuronal simulations and generate benchmark output.

`beNNch` is installed using `Builder` (Computational and Systems Neuroscience & Theoretical Neuroscience 2021b). `Builder` ensures machine specific MPI implementations as well as other libraries are provided during installation. `beNNch` employs the xml-based `JUBE` Benchmarking Environment (Centre

2021) using its `yaml` interface. `JUBE` creates (`SLURM`) job scripts, submits them, and gathers and unifies the raw data output. The `yaml` interface provides a user-friendly configuration script where users can specify amongst others, the network being simulated, scaling type, number of nodes, number of threads, and simulator type and version. The variables which appear on the benchmark output may also be configured to include only results the researcher is interested in. The timing information contained in the output can either come from the built-in `NEST`'s C++ level timer, or from `PyNEST`'s Python level timer. The Python timer called `wall_time` is realized through an explicit call to `time.time()`. It represents the actual time that phase of simulation takes. `NEST`'s built-in timer called `py_time`, on the other hand, provides a detailed look into the contribution of all four phases of state propagation[4]. We choose the `wall_time` for our experiments as we want the actual time of simulation.

2.3.3 Analysis and visualization software

Intel VTune Profiler[5] is a parallel performance analyzer tool which is used in low-level performance analysis to study the behavior of multithreaded software. It determines the functions and areas where the most time is taken; sections of the code that do not utilize resources; as well as thread activities and hardware-related issues. It is used in this thesis to analyze the performance of some of the networks using different numbers of MPI processes and threads configuration combinations. For visualization and generation of the plots and some of the tables in this thesis, the following software tools are used: Pandas[6] for data analysis and managing the data in a structured way; Seaborn[7]; and Matplotlib[8] for plotting graphs.

2.4 Balanced random networks

All the benchmarks performed in this thesis are performed by constructing and simulating variants of the widely-used balanced random network model, (inspired by the Brunel model (Brunel 2000)). The base network consists of 65,000 n_E (excitatory neurons) and $65000/4 = 16250$ n_I (inhibitory neurons). Making a total of 81250 neurons at the start, i.e. $n_{SCALING} = 1.0$. During simulation the total number of neurons (N) is scaled to adjust network size during weak scaling. All the connections exhibit no plasticity (i.e. static synapses). The network dynamic is relatively simple mainly because the focus

[4]The four stages of propagation are delivery, communication, collocation and update.
[5]https://www.intel.com/content/www/us/en/develop/documentation.html
[6]https://pandas.pydata.org/
[7]https://seaborn.pydata.org/
[8]https://matplotlib.org/

of this thesis is at the network construction phase. We use the spiking data as a check for correctness.

The neuronal parameters as well as network parameters are summarized in Table 2.5. The potential difference is set at $0.0mV$ due to the ease with which it will bring to calculate with 0 as the reference point. All neurons receive stationary external input in the form of Poisson spike trains with fixed rate parameters. The number of excitatory synapses per neuron (c_E) increases up until a cap at 6500 and 1500 for the number of inhibitory synapses per neuron (c_I) no matter how large the network gets. This ensures the total number of connections created does not grow too rapidly.

Networks with two types of architectures based on their spatial dependence were created for this thesis, They are: Spatially distributed networks and non-spatially distributed networks. All the spatial networks are setup using pairwise Bernoulli connection rules. They all have circular masks with a radius of 0.5 from which the source neurons are selected. Autapses are permitted here. The spatial networks differ in the type of probability function passed in the `Connect()` command. In this thesis, four different probability types are used namely:

- `Gaussian`,
- `Gaussian_tab`,
- `Gaussian_Ex`,
- `Circular`.

The spatial networks are hence named after these probability functions and shall be called as so henceforth. All except Circular are Gaussian shaped functions implemented in different ways. The Gaussian function can be seen in Table 2.4. The Gaussian network's probability is implemented using the `NEST` builtin function for spatial distributions. A standard deviation of 0.14 is passed in together with the distance between the source and target neurons. The `Gaussian_tab` network is a network with a Gaussian function which is calculated by picking up the exponentials from a pre-filled exponential table and then estimating the closest exponential based on this value. This exponential is then used in computing the Gaussian value. The `Gaussian_Ex` network has its probability function calculated explicitly at the interpreter level. The Circular network has a fixed probability value of 0.15655 for every connection. As a precautionary step, the reference network model, the `Gaussian` network is also created on another version of NEST, the older NEST v3.1 (Deepu et al. 2021).

Some of the connection rules are shown in Listing 1. The spatially distributed network had nodes arranged into layers which are `NodeCollections` with spatial metadata. The nodes are placed freely in space (i.e., arbitrarily in

```python
# GAUSSIAN
probability = nest.spatial_distributions.gaussian(nest.spatial.distance,
                                                  std=0.14)
# GAUSSIAN EXPLICIT
probability = nest.math.exp(-0.5 * (nest.spatial.distance/0.14)**2)
# CIRCULAR
probability = 0.15655
conn_rule = {
    'rule': 'pairwise_bernoulli',
    'p': probability,
    'mask': {'circular': {'radius': 0.5}},
    'allow_autapses': True
}
```

Listing 1: Spatial network connection . Snippet of code used to form connection rules in the spatial networks.

the plane) randomly within the *extent*. Our spatial networks span a square of $[-0.5, 0.5] * [-0.5, 0.5]$. The extent is set as $1 * 1$, meaning it spans the entire square. During weak scaling, the square size increases by a multiple of the square root of the $N_{scaling}$. A periodic boundary condition is set in order to reduce the effect of boundaries on simulations so that the nodes at opposite edges are considered as nearest neighbors (torus connectivity).

A mask is specified to map out the potential sources. The mask size has to be smaller than the layer size. A circular mask with a radius of 0.5 is specified as the mask. The spatial metric used during connection is the distance between the source and target nodes. For calculating the random distribution based on the position of the nodes, the following functions are used: Gaussian distribution,

$$p(x) = e^{-\frac{(x-mean)^2}{2std^2}} . \tag{2.5}$$

Where $p(x)$ is the probability function, *std* is the corresponding standard deviation defining the spatial width of the profile, and x is the population. The Gaussian distribution is used as the reference network when comparing the total number of connections formed. The Gaussian-Explicit distribution is the explicit calculation of the Gaussian function from the Python (interpreter) level. The exponential is calculated using `nest.math.exp`. For the circular distribution, a constant probability is used in making all connections. The Gaussian-Tabulated function is calculated using a lookup table. This lookup table function was implemented during this thesis in order to reduce the total amount of exponential calls made. Listing 2 shows how the exponentials are calculated and added to the exponential table. Listing 3 shows the lookup

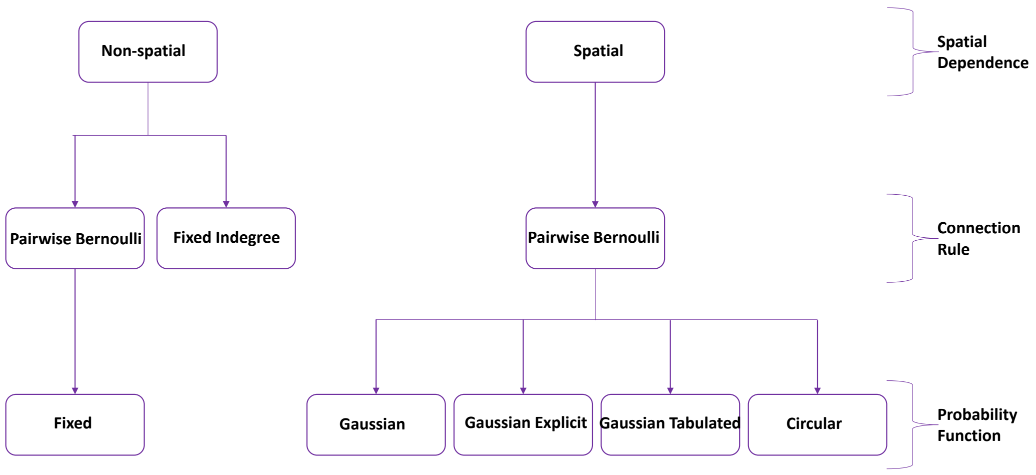

Figure 2.5: Overview of the network types. The networks are divided by their spatial dependence, then the connection rules which they have, and finally the probability function (except for fixed indegree which has no probability function).

process using linear interpolation. Figure 2.6 show the masks for fixed and Gaussian probabilities.

For the non-spatial networks, two types of connection rules are used to make the connections depending on the type of network. They are fixed indegree and pair wise Bernoulli. These are setup in the `rule` parameter using the entries:

- `fixed_indegree`,
- `pairwise_bernoulli`.

The `"fixed indegree"` network, named after its connection rule; and the `"nonspatial fixed"` network, named such because it has a pairwise Bernoulli connection with a fixed probability value of 0.11511. In fixed indegree, the source nodes are randomly connected with the target nodes such that each of the target node has a connection to a fixed number of source nodes. The fixed indegree network has an indegree value of 9990. The values of their probabilities (and indegree) were gotten after comparing the number of connections generated by each network to that of the Gaussian network — which is our reference network. Table 2.2 shows the ratio of the number of connections formed by every network compared to that formed by the Gaussian network. After the simulator receives these commands it iterates over source and target neurons, creates pairs based on the probability value (or indegree) and connects the pairs. Figure 2.5 give an overview of connectivity and probability functions of the networks.

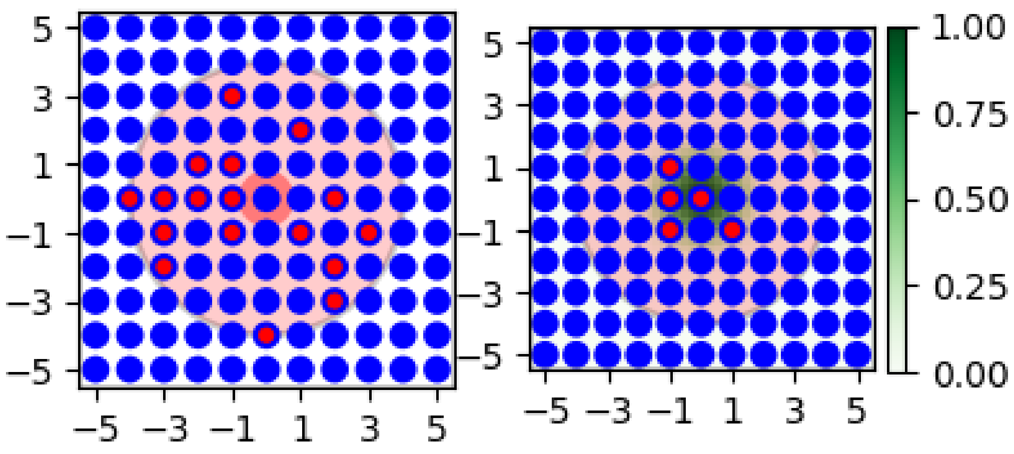

Figure 2.6: Illustration of circular and Gaussian connection probabilities (source: NEST documentation). Top left has constant probability of 0.5. Top right: Distance dependent Gaussian probability, green distribution show variance.

In pairwise Bernoulli, a connection is made between every possible pair of source and target nodes with a user defined probability value, p. For connections in the pairwise Bernoulli, we limit the source nodes to a fraction of the total nodes by an area with a radius of 0.5. That is,

$$n_{source} = n_{population} \times \pi \times r^2 \ . \tag{2.6}$$

Here n_{source} is the number of neurons which will be selected as the source neurons, $n_{population}$ is the population whose nodes are being selected. This is either a population of excitatory or inhibitory nodes, and r is the radius of the source layer/mask. This is important because in weak scaling the population is multiplied by the number of nodes in order to increase the size of the problem as the number of compute nodes increase. This means the number of connections formed will also grow very rapidly. Thus, during weak scaling the growth of the number of connections is curbed. Connections are made with a `Connect()` call. Listing Every network is comparable as the number of connections as a ratio to the reference network, the spatial Gaussian network, has a value of approximately 1.0 to the nearest 4 decimal places. Figure 2.5 shows an overview of all the networks used in the thesis by the spatial dependence and connection rules/probability distribution.

Afterwards, `Prepare()` is called to calibrate nest before calling the function which simulates the network for t milliseconds, `Run()`. The simulation time is set to $t = 100.0$ms. Finally `Cleanup()` is called to close files and cleanup handles. The memory and time taken during the initialization stage, network creation stage, connection stage and simulation stage are recorded and written to a file. The initialization stage is the stage where the kernel is reset and

```cpp
for ( size_t i = 0; i < size_t(std::ceil( table_max_/table_step_ + 0.5) );
      i++){
      const auto dx_table = table_step_ * i;
      table_values_.push_back( std::exp(
      -dx_table * dx_table * inv_two_std2_ ) );
}
```

Listing 2: Exponential calculation . The function loops through for `table_max`/`table_step` times and inserts the exponentials. The `table_max` and `table_step` are provided by the user during the network connection creation phase.

initialized with some simulation parameters. The creation stage is where the nodes (neurons and Poisson generators) are created. The connection stage begins with creation of edges (synapses) between neurons and stops after the `Prepare()` command which is called when the Poisson generators and neurons have been connected. The simulation stage is where the call to `Run()` and `Cleanup()` occurs. Tables 2.3, 2.4 and 2.5 provide an overview of the balanced random network used in this thesis.

2.5 Benchmark Protocol

In this section we discuss how the benchmarks are obtained. Thus far in this chapter, we have presented the different materials used in this thesis. At a high level, we obtain the quantitative results obtained here by constructing and simulating the various network models (Section 2.4) with specifications in Tables 2.3, 2.4 and 2.5 on a supercomputer with specifications in Table 2.1. To attain this, we create a variant of the balanced random network (Brunel 2000) in NEST (Spreizer et al. 2022). The network is created at the Interpreter level with a leaky integrate-and-fire (LIF) neuron model (Lapique 1907) as the base neuron model, and a non-plastic synapse model as the base synapse model. The neuronal parameters are set as defined in Table 2.5.

NEST's kernel is reset and set up with some simulation parameters before creating the network. These parameters are the local number of threads per MPI process, resolution of the simulation, and the seeds to use for reproducibility. We use two seeds in this thesis. One seed is used for running simulations multiple times, and the other seed is used for verification should we get any non-conforming results (i.e., results which do not conform to scalability theories or other relevant theories). The network is afterwards initialized by creating the excitatory and inhibitory populations using one `Create()` command for each population. The neuronal parameters are passed into this command also. A position parameter is also passed for the spatial networks. This position

```cpp
const auto dx = p_->value( rng, source_pos,
                           target_pos, layer, node ) - mean_;
size_t left = std::floor( dx / table_step_ );
size_t right = std::ceil( dx / table_step_ );

double relative_position_in_interval = (
  dx - left * table_step_ ) / table_step_;
double interpolated_value = table_values_.at(left) + (
  table_values_.at(right)-table_values_.at(left)) *
  relative_position_in_interval;

return interpolated_value;
```

Listing 3: Lookup function . The function performs a linear interpolation and estimates the nearest exponential.

parameter has a free layer so the nodes' positions are not restricted to a grid. It is also set up with a periodic boundary condition giving it a torus shape such that rightmost and topmost elements have the leftmost and bottommost elements respectively as their nearest neighbors, effectively reducing the effect of boundaries on simulations. This is specified using the entry `edge_wrap`.

External input from a Poisson generator is also created with a rate

$$rate = \frac{1000.0 \times eta \times V_{th} \times C_m}{J_{ex} \times C_E^2 \times \exp 1 \times \tau_m \times \tau_{syn}} \, . \tag{2.7}$$

Where $rate$ is the mean firing rate, eta is the external rate relative to threshold rate, V_{th} is the threshold potential of the neuron, C_m is the membrane capacitance, J_{ex} is the amplitude, calculated by $\frac{J}{J_{unit}}$. J is the post synaptic amplitude, J_{unit} is the normalized postsynaptic current calculated for one unit of amplitude. C_E is the max indegree of the excitatory nodes, the rest of the parameters are as specified in Table 2.5. Afterwards, recurrent connections among the neurons are established. The excitatory and inhibitory synapses are setup as static synapse models with fixed weights J_{ex} and $J_{in} = -g * J_{ex}$ (g is the ratio of inhibitory to excitatory weights) for excitatory and inhibitory synapses respectively, and a delay of 1.5. After setting up the synapses, the Poisson generators are connected to the populations. The final phase is the creation of connections between the neurons using the synapses.

All of the aforementioned steps are collectively part of the benchmarking work done in this thesis. Collation of all the performance data is also done afterwards; these two processes constituting the benchmarking done in this thesis. `beNNch` is used to run the benchmarks using a number of `YAML` scripts to pass the simulation of the network into the JUSUF supercomputer. The

```
p_fixed = 0.11511
indeg = 9990
excit_ratio = 0.8 # 0.8 for excitatory, 0.2 for inhibitory
# INDEGREE
conn_rule = {'rule': 'fixed_indegree',
             'indegree': int(excit_ratio * indeg),
             'allow_autapses': True
}
# FIXED PROBABILITY
conn_rule = {'rule': 'pairwise_bernoulli',
             'p': probability,
             'allow_autapses': True
}
```

Listing 4: Non-spatial network connection . Snippet of code used to make connection rules in the non-spatial networks.

simulations are run 5 times with the same seed to check the variability of the data. A number of different hardware configurations and the networks are provided to `beNNch` in one of the scripts. `beNNch` uses JUBE for job management. Our output is converted to a number of comma separated value (CSV) files for analysis. From the analysis, a number of plots were generated to provide visual aid. The reference model is also run on Vtune to get performance information. The analysis of the benchmarks are discussed in detail in Chapter 3.

Number of MPI Processes	Gaussian vs Gaussian	Gaussian vs Gaussian Explicit	Gaussian vs Circular	Gaussian vs Gaussian Tabulated	Gaussian vs Fixed Indegree	Gaussian vs Fixed Bernoulli	Gaussian vs Gaussian NEST 3.1
1	1.0	1.0	0.999969	1.000416	1.000021	0.999958	0.999982
2	1.0	1.0	1.000006	1.000416	1.000039	0.999996	1.000043
4	1.0	1.0	0.999985	1.000416	1.000013	0.999978	0.999999
8	1.0	1.0	1.000001	1.000416	1.000006	0.999974	0.999986
16	1.0	1.0	1.000005	1.000416	1.000002	0.999974	0.999980
32	1.0	1.0	1.000009	1.000416	0.999998	0.999973	0.999994
64	1.0	1.0	1.000016	1.000416	1.000006	0.999982	1.000012

Table 2.2: Ratio of number of connections formed for all networks to Gaussian network

A	Model Summary
Populations	Three: excitatory, inhibitory, external input
Topology	Spatial (periodic boundary condition); Non-spatial
Connectivity	Pairwise Bernoulli; Fixed Indegree
Neuron model	Leaky integrate-and-fire, fixed voltage threshold, fixed absolute refractory time (voltage clamp)
Synapse model	Static synapse

B	Populations	
Name	**Elements**	**Size**
E	Iaf neuron	$N_{\mathsf{E}} = 4N_{\mathsf{I}}$
I	Iaf neuron	N_{I}
$\mathsf{E}_{\mathsf{ext}}$	Poisson generator	$C_E(N_{\mathsf{E}} + N_{\mathsf{I}})$

C1	Connectivity - Fixed Indegree		
Indegree (non-spatial) = 9990			
Name	**Source**	**Target**	**Pattern**
EE	E	E	random independent connections, fixed indegree C_{E}, weight J_{ex}, uniformly distributed delay values
IE	E	I	random independent connections, fixed indegree C_{E}, weight J_{ex}, uniformly distributed delay values
EI	I	E	random independent connections, fixed indegree rule, weight J_{in}, uniformly distributed delay values
II	I	I	random independent connections, fixed indegree rule, weight J_{in}, uniformly distributed delay values
Ext	$\mathsf{E}_{\mathsf{ext}}$	E ∪ I	Non-overlapping $C_{\mathsf{E}} \to 1$, weight J_{ex}, delay D

Table 2.3: Description of balanced random network model following the guidelines of (Nordlie et al. 2009). Distinction between spatial and non-spatial networks. **A** gives the entire summary of the model, here, the two types of topologies are shown as well as the two types of connectivities. **B** gives a summary of the populations of nodes. **C1** gives the connectivity patterns for the non spatial fixed indegree connections.

C2	Connectivity - Pairwise Bernoulli		
Gaussian-shaped		Fixed	
$p(x) = e^{-\frac{(x-mean)^2}{2std^2}}$		Circular mask (spatial), $p(x) = 0.15655$ Fixed (non-spatial), $p(x) = 0.11511$	
Name	**Source**	**Target**	**Pattern**
EE	E	E	random independent connections, pairwise bernoulli rule, weight J_{ex}, uniformly distributed delay values
IE	E	I	random independent connections, pairwise bernoulli rule, weight J_{ex}, uniformly distributed delay values
EI	I	E	random independent connections, pairwise bernoulli rule, weight J_{in}, uniformly distributed delay values
II	I	I	random independent connections, pairwise bernoulli rule, weight J_{in}, uniformly distributed delay values
Ext	E_{ext}	$E \cup I$	Non-overlapping $C_E \rightarrow 1$, weight J_{ex}, delay D

D	Neuron and Synapse Model
Name	Iaf neuron
Type	Leaky integrate-and-fire, δ-current input
Sub-threshold dynamics	$\tau\dot{V}(t) = -V(t) + RI(t)$ if $t > t^* + \tau_{\mathrm{rp}}$ $V(t) = V_{\mathrm{r}}$ else $I(t) = \frac{\tau}{R}\sum_{\tilde{t}} w\delta(t - (\tilde{t} + \Delta))$
Spiking	If $V(t-) < \theta \wedge V(t+) \geq \theta$ • emit spike

Table 2.4: Continuation of Table 2.3. **C2** gives the connectivity patterns for networks based on Pairwise Bernoulli connections. Both spatial dependent and non-spatial dependent networks connect this way. For the Gaussian shaped distributions, only the spatial networks are implemented with that shape. The probability function is given. For fixed on the other hand, there is one spatial network which has a fixed probability with a circular mask. The non spatial one is also given. **D** summarizes the neuron and synapse model.

E	Input
Type	**Description**
Poisson generators	Fixed rate ν_{ext}, C_{E} generators per neuron, each generator projects to one neuron

F		Parameters
Name	**Value**	**Description**
N_E	65000	Number of excitatory neurons
N_I	16250	Number of inhibitory neurons
$C_{E(max)}$	6500	Max number of incoming excitatory neurons
$C_{I(max)}$	1500	Max number of incoming inhibitory neurons
V_m	$0.0mV$	Membrane potential average
E_L	$0.0mV$	Reset membrane potential of the neurons
V_{th}	$20.0mV$	Threshold potential of the neurons
V_{reset}	$0.0mV$	Membrane potential after a spike
C_m	$250.0pF$	Membrane capacitance
τ_m	$20.0ms$	Membrane time constant
τ_{syn}	$0.5ms$	Time constant of postsynaptic currents
τ_{ref}	$2.0ms$	Refractory period of the neurons after a spike

Table 2.5: Continuation of Table 2.4. **E** Shows the input from the Poisson generators. **F** Gives some of the important neuronal parameters used in the networks.

Chapter 3

Results

In this chapter, we present the results of our simulation experiments, as well as the findings from the analysis of the scaling performance of the different neuronal networks used. For this study, we use the Circular network model implemented in the newest version of NEST, version 3.3 (as at the time of writing this thesis), as the reference model for the spatial networks. We use the Connecting time in seconds as the metric for measuring the performance of the model. The connecting time is the wall time of a given simulation. The number of MPI processes is the compute resource used in the scaling experiments. Each of the different simulations have been run on different thread combinations.

3.1 Cross-version verification

In order to verify our results are consistent across other stable NEST versions and have similar behavior for the same network of similar connection sizes, we compare the simulation of one network in 2 different NEST versions. We chose to compare the weak-scaling experiments of the Gaussian model implemented in the latest NEST version 3.3 with the simulation of the same model implemented in NEST version 3.1. As NEST 3.3 has the some backward compatibility, it was just a few changes that were needed to be made to make the same networks simulate on both versions of NEST. Since we use the `beNNch` framework for benchmarking, the simulation results format should not be very different.

The result of the comparison is shown in Figure 3.1. We observe that both networks have very similar wall times in this experiment. There is an obvious improvement in performance in NEST 3.3 most probably owing to the improvements in parallel network construction time if large numbers of devices are present due to accelerated node lookups[1] which has been shown to reduce

[1] https://github.com/nest/nest-simulator/pull/2290

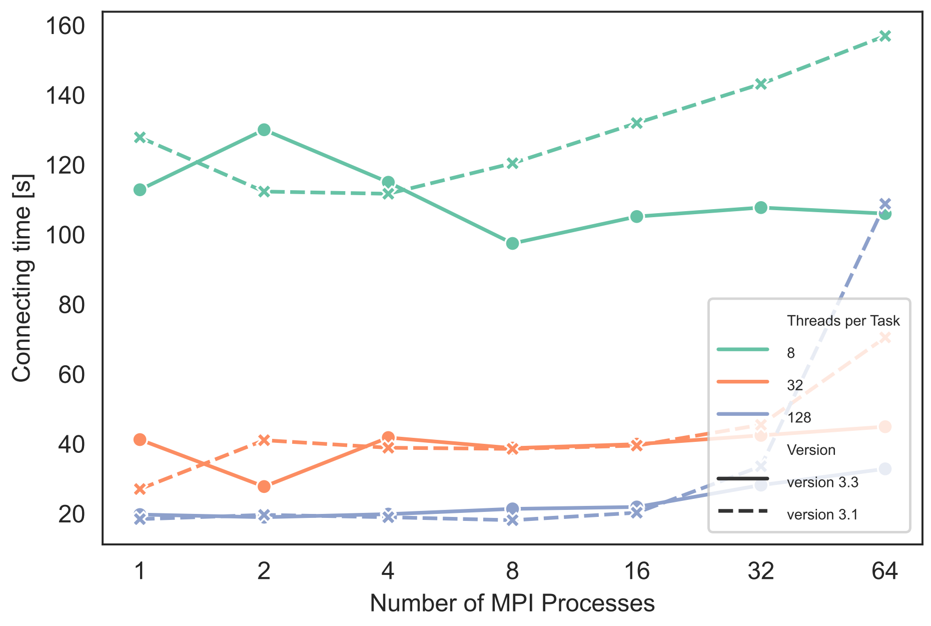

Figure 3.1: Weak scaling of the Gaussian networks for NEST versions 3.3 and 3.1. The wall time taken to construct connection between the neurons on different combinations of compute nodes (MPI processes) for different number of threads used per task is shown in the figure. The wall time for connection in seconds is shown on the vertical y axis while the number of compute nodes used per experiment is shown on the horizontal axis. In the plots, the different lines are the results for running the simulation on different compute nodes using different threads per task. The colors and solidness of the lines shown in the legend correspond to the threads per task. The dashed lines show the weak scaling of the simulation of the Gaussian network on NESTv3.1, while the solid lines show the weak scaling of the simulation of the Gaussian network on NESTv3.3.

network construction time by a factor up to 20 in some specific models (Spreizer et al. 2022). It can be seen that for smaller threads per tasks, the weak scaling performance is relatively close to the ideal performance. From larger threads the performance starts to deviate away from the ideal performance. Up to 16 MPI processes, the two benchmarks can be said to exhibit similar scaled speedup. This result suggests that the simulation of our reference network on NEST is veracious.

3.2 Weak scaling performance for non-spatial networks

In this section we assess the weak-scaling performance of the non-spatial networks simulated on NEST v3.3. The minimum `connection time` for connec-

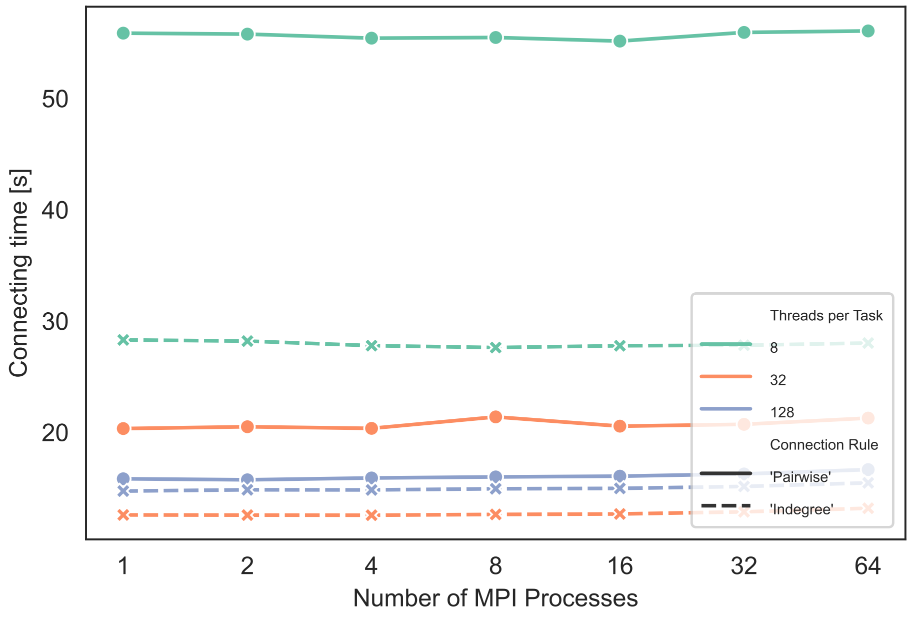

Figure 3.2: Weak scaling of the Non spatial networks for Indegree and Bernoulli pairwise connection rules. The solid lines indicate the performance of the Pairwise nonspatial network, while the dashed line represents that of the Indegree. The various numbers of threads used are represented by the colors in the legend.

tion construction of the scaled various networks after five runs is studied. The networks are scaled by the MPI process as described in the Methods. In Figure 3.2, we can see the performance of the non-spatial network with indegree connection rules against that of the pairwise-Bernoulli rule. The plot shows that the scaling experiments have a more constant time. The constant time implies that the weak scaling of an experiment is close to ideal. In the case of the indegree network, it performs better than the pairwise network for the same number of threads. Since the number of connections formed has been verified; this implies that the time it takes the NEST simulator to create a connection with a pairwise Bernoulli rule is significantly greater than one where the number of incoming connections per target neuron is already set. This performance difference is expected as in the Pairwise Bernoulli networks, an iteration across all the nodes is required before forming connections since it is not known beforehand how many connections will be made, and whether connections will be made. As it is a fixed probability, the obvious reason for the difference in the performance across both networks will be the iteration.

In the Bernoulli pairwise network, the lower the number of threads, the more time it takes. Meanwhile, for the indegree case, the lower the number of threads, it is not clear. As can be seen, the case of the 128 threads take more

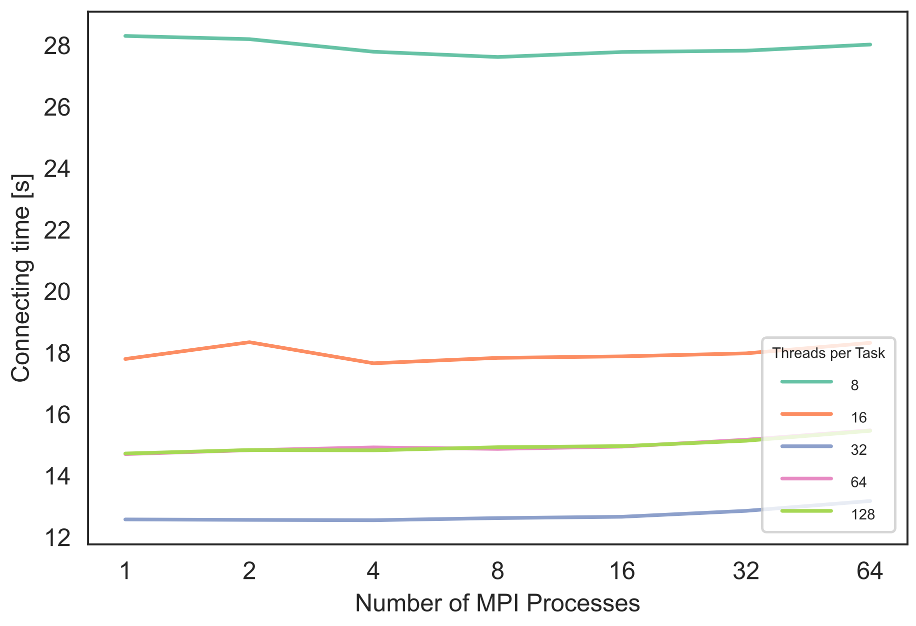

Figure 3.3: Weak scaling of the Indegree network. The various numbers of threads used are represented by the colors in the legend.

time than that of the 32 threads. To properly verify this, we check Figure 3.3. Here, you can see that from 8 threads up to 32 threads, the time reduces significantly as is to be expected. From 64 and including 128 however, the time climbs back up, performing worse than the 32 threads per task. This suggests that around 32 threads is the optimal number of threads for connecting with indegree connection rules. It is interesting that the wall time difference between 64 and 128 threads is almost indistinguishable. This will most likely signify that with additional threads above 32, the performance reduces and adding even more threads will not cause any significant change in the performance. From 32 MPI processes to 64 MPI processes, a slight deviation from the ideal scaled speedup begins to happen, and is expected to continue for larger MPI processes beyond 64 MPI processes. This would mean that increasing MPI processes beyond 64 nodes would not offer the same ideal parallelism as with smaller MPI processes.

3.3 Weak scaling performance for spatial networks

In this section, we discuss the performance of the spatial networks used in this paper. We start by examining our reference model, the Circular model, and

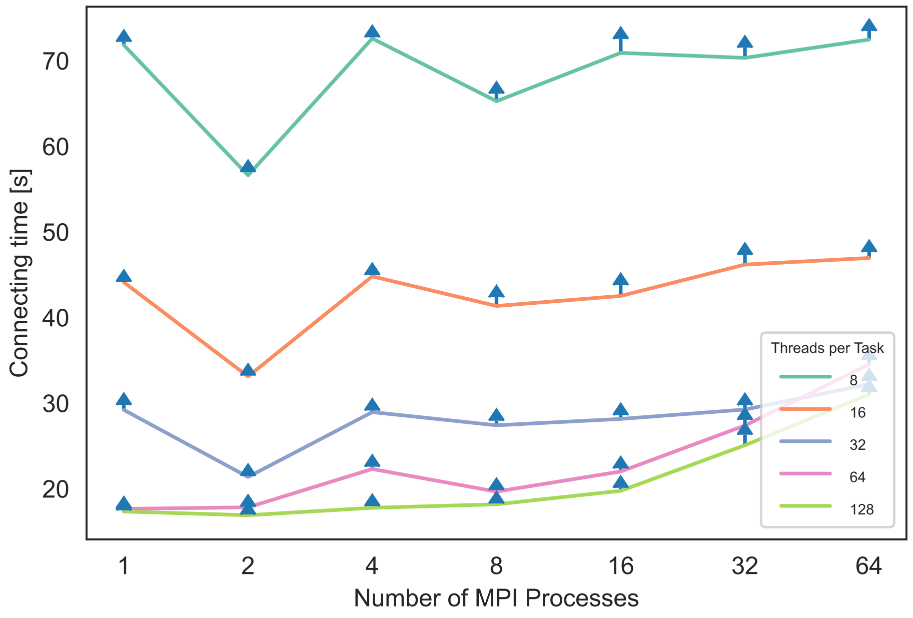

Figure 3.4: Weak scaling of the Circular network. The various numbers of threads used are represented by the colors in the legend. The blue arrows represent the standard deviation across various runs.

then compare this model to a Gaussian model. Like the nonspatial case, the models discussed here are simulated on NEST v3.3. The minimum `connection time` for connection construction of the scaled various networks after five runs is studied. The Gaussian network model, as indicated in Chapter 2, is our reference network model. Figure 3.4 shows that performance of this network is not as good as the Indegree network, both in connecting time and in weak scaling speedup. Here, you can see that there is a huge performance increase for 2 MPI processes from 8 threads to 32 threads. The reason for this performance increase is unknown. We verified if this was the case using 3 different seeds but got the same performance increase at the same points. This strange performance was investigated further by checking the hardware setup at that point, and the simulation metadata. Nothing signified any anomalies in the hardware system. The same pattern is noted across all spatial networks used in this paper. The deviations represented by the blue arrows show that the 5 runs are similar in their connecting times.

Comparing the Circular network to the simplest Gaussian network as can be seen in Figure 3.5, the Circular network has significantly lower connection creation times than the Gaussian network model. For the 8 threads down to the 128 threads, there is a significant reduction in the difference in their connecting times. For 128 threads, they are both around 20 seconds. Aside

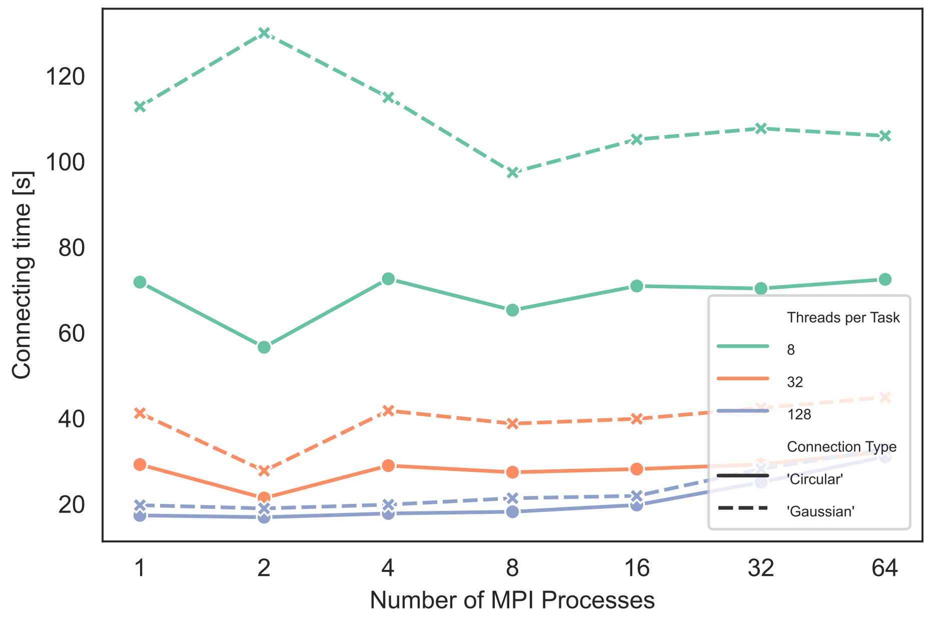

Figure 3.5: Weak scaling of the spatial networks for Gaussian and Circular network models. The solid lines indicate the performance of the Circular network, while the dashed line represents for that of the Gaussian. The various numbers of threads used are represented by the colors in the legend.

from the connecting time changes around where it is 2 MPI processes, the increase in number of MPI processes is roughly constant. With larger threads, it is less constant, and starts to rise slightly. This shows that the performance increase with respect to weak scaling by increasing the number of threads gets lower and lower, signifying that the network is not utilizing the threads as efficiently as with less.

The reason as to why the non-spatial Bernoulli network has a significantly better connecting time than the circular network despite them both having fixed probabilities and having the same pairwise Bernoulli rule points to the complexity of spatial versus non-spatial connection creation. For the Circular network, connections are made depending on the relative positions of source and target neurons with a fixed probability using a circular mask. For the difference in connecting time between the Circular network and the Gaussian network, it points to the implementation of the more complex Gaussian equation, $p(x) = e^{-\frac{(x-mean)^2}{2std^2}}$, in the simulator. In the Gaussian not all nodes are connected, typically, a reduced chance of connection the farther away the node is. The computation of the exponential is usually very expensive when performed several thousand times.

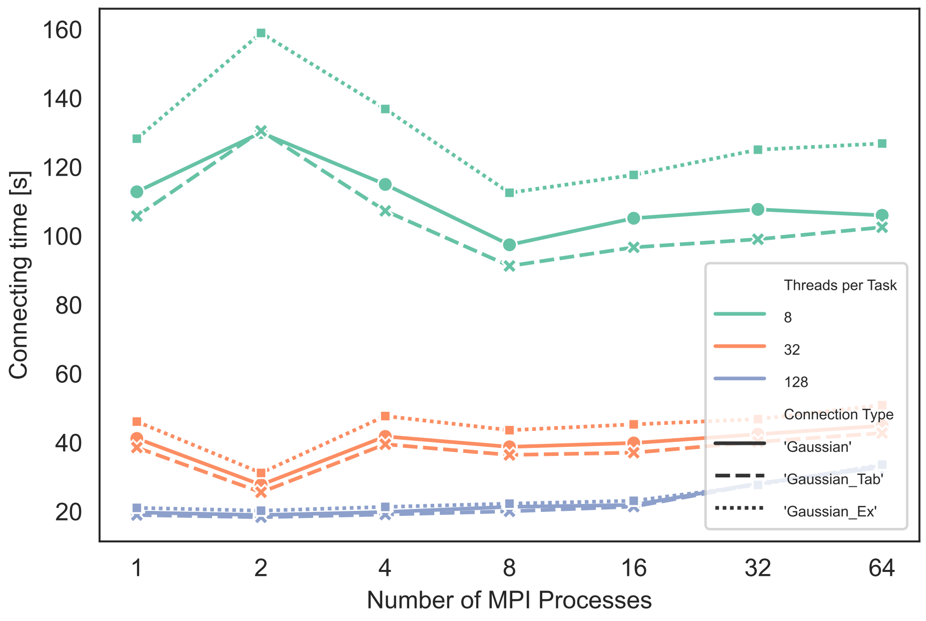

Figure 3.6: Weak scaling of the spatial networks for Gaussian, Gaussian Explicit, and Gaussian Tabulated network models. The solid lines indicate the performance of the Gaussian network, the dotted lines represent the Gaussian Explicit network, while the dashed line represents for that of the Gaussian Tabulated. The various numbers of threads used are represented by the colors in the legend.

3.3.1 Spatial Gaussian network model analysis

To verify the performance of NEST in computing the Gaussian exponential function, we compare three different Gaussian based networks:

- A Gaussian network with a probability distribution provided by:
  - `nest.spatial_distribution.gaussian()`;
- a Gaussian Explicit network with its probability distribution explicitly defined by:
  - `nest.math.exp(-0.5 * (nest.spatial.distance/std)**2)`.
  `nest.math.exp` is how you explicitly call NEST's exponential formula. The rest of the code is as in the Gaussian equation;
- a Gaussian Tabulated network with the call:
  - `nest.spatial_distribution.gaussian_tab()`;

The Gaussian Tabulated network probability function was added during the course of writing this thesis. The steps taken and code listings are described in Chapter 2 in Listing 3. This function was implemented in NEST v3.3 revision 7788ee. Figure 3.6 show the comparison of the performance of these networks in the weak scaling experiment. The Gaussian Explicit network consumed the most time in connecting. The Gaussian and Gaussian Tabulated networks consume roughly similar connecting time with the Gaussian Tabulated network edging the Gaussian network by a few seconds as can be seen in the figure. With increased numbers of threads, this difference becomes less prominent until they almost even out at 128 threads per task. The scaling patterns across all Gaussian networks are largely similar. The reason why the Gaussian explicit takes significantly more time to create connections is largely because the explicit function is done at the Python level as opposed to the other networks where their explicit functions are done at NEST's C++ kernel level.

3.4 Strong scaling performance overview

This section describes the result of the strong scaling experiment of some of the networks. In here, the metric used to measure performance is the scaled connecting time. This was gotten by multiplying the final result of the connecting time after simulation by the number of MPI processes. This way, the scaled connecting time would be expected to have constant time during scaling across MPI processes. The result will also be comparable with the weak scaling result. Figure 3.7 show the illustration of the comparisons between the strong scaling of the Gaussian, Circular and Indegree networks. The Indegree network exhibits very close to constant time strong scaling, meaning it follows the ideal pattern for good strong scaling. The circular network on the other hand isn't a very straight line. For 2 MPI processes, there is the same sudden reduction of connecting time as seen in its weak scaling for 8 and 32 threads. For 4 MPI processes, like in weak scaling, there is also a sharp increase in (scaled) connecting time from around 60 seconds to around 85 seconds in the case of 8 threads. This same pattern is also seen for 32 threads. With 8 threads, scaling from 1 MPI process to 64 processes, there is a gradual reduction in scaled connection time. For 32 and 128 threads per tasks, going from 1 MPI process to 64 MPI process, there is a gradual increase from around 19 and 30 seconds to about 80 and 50 seconds respectively. This same pattern noticed in the Circular occurs in the Gaussian network as well.

The poor strong scaling in the spatial networks across the bigger number of threads show that for a fixed sized spatial networking, adding resources in terms of threads and MPI processes does not improve the scalability, rather, it causes more overhead to the system. In terms of scaled connecting times along the number of threads. The scaled connecting time of the Gaussian and Circular is just as it was in ranking as with that of the weak scaling results. The indegree network, as in weak scaling, performed in the shortest

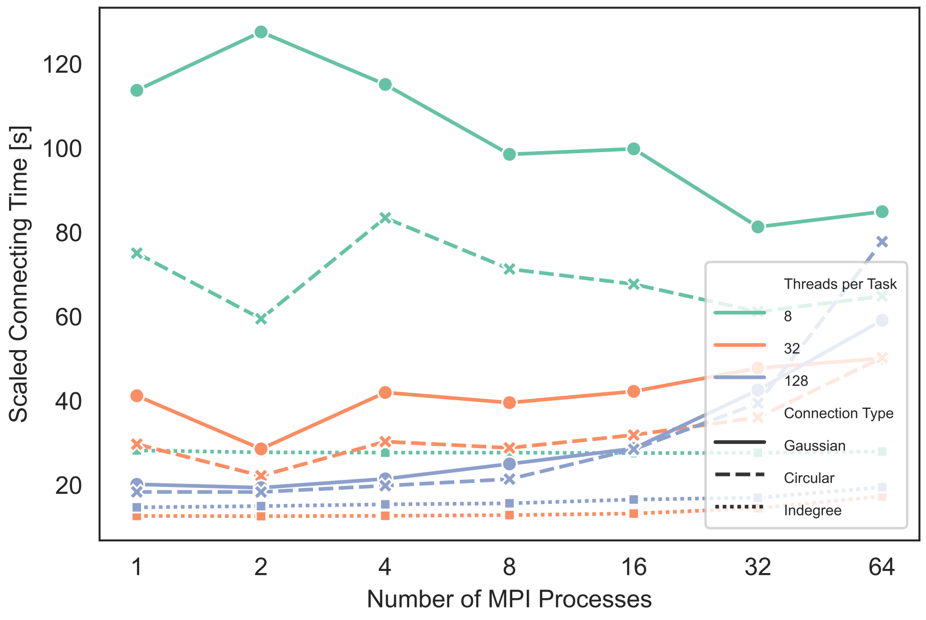

Figure 3.7: Strong scaling of Gaussian, Circular, and Indegree network models. The solid lines indicate the performance of the Gaussian network, the dotted lines represent the Indegree network, while the dashed line represents for that of the Circular network. The various numbers of threads used are represented by the colors in the legend.

connection times across all networks. Figure 3.8 shows that although the connection times for every thread per task is roughly similar, the weak scaling experiment performs better. The same pattern with an increase in connection time with any increase from 32 threads, repeat here.

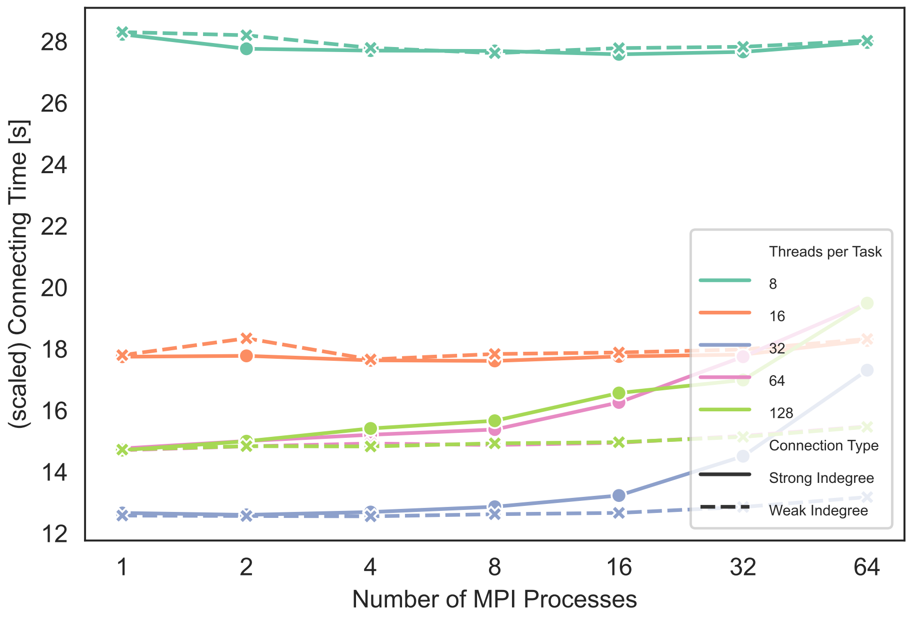

Figure 3.8: Strong and weak scaling of the Indegree network model. The solid lines indicate the strong scaling performance of the non-spatial Indegree network while the dashed lines represent that of the weak scaling. The various numbers of threads used are represented by the colors in the legend.

Chapter 4

Discussion

We have presented our findings from the weak and strong scale experiments of different networks. In this chapter, we provide reasons as to why some of the results presented in Chapter 3 are the way they are. Some more plots providing further insights into the performance of some of these networks are provided. We also discuss our findings from the VTune profiling performed. We focused on connection creation phase in this paper, therefore, we tried to keep all of the network dynamics as simple as possible while also capturing the following properties: the spatial dependence and the probability function. These different properties hugely alter the connection rules in networks. For spatial dependence, it brings a new dimension of distance to the source node (depending on connection rule), as well as the representation in the simulation. In the probability function, it affects how much time the simulator takes when making connections.

In the Indegree connection as seen in the results presented in the previous chapter, due to the fact that the number of connections each source neuron will form with the target neurons is already known, the simulator does not have to go through every possible target neuron. Since NEST provides a shuffled index of all the neurons, it simply selects to the amount of connections it is supposed to make and then the next source neuron does the same till every connection is made. In the case of the pairwise Bernoulli non spatial network, it can be seen that the connection time is higher than the fixed indegree (shown in Figure 4.1). This ranking is based on the fastest to the slowest networks to connect. This is owing to the fact that it has to check every node along the target population before making connections with a fixed probability.

For the spatial networks, they all take more to create connections than the non spatial networks. An inspection using Intel VTune Profiler suggested that a huge chunk of time taken for the connection construction occurs during searching/ sorting. One possible way to improve on this will be finding a different representation of the source and target population in the simulator. This difference is to be expected however. One point of interest we noticed during this project was that in the connection times for 2 MPI processes, there was a sudden dip across most of the threads per task. The reason behind this phenomenon was not obtained despite using different seeds and inspecting the hardware configuration and metadata during the simulations. The subtractions between closely ranked networks suggest that as we go towards using more threads, the performance loss between the different closely ranking networks move towards zero. This can be seen in Figure 4.2.

The Gaussian Tabulated network which was implemented in this paper shows that there is a slight improvement when using a table of values to interpolate the exponential values. The difference in performance is still some ways away from the circular network. This can be improved however, working on different step values to find if it is worth including into a stable NEST version in the future. In Figure 4.3, we make a plot of virtual processes against connection time. The virtual processes is gotten from multiplying the MPI processes with the number of threads. This provides a good overview of the speed up achieved with different MPI processes. We can clearly see that the speed up shows that for the non-spatial, there is very little overhead with respect to resources as against the spatial one which has somewhat higher over head both for number of MPI processes and number of threads. It must be noted this overhead increases when the virtual processes increase. There is therefore good gain in speed up, but adding more resources past around 250 virtual processes will not improve the speedup, but will eventually cause a reduction in the performance.

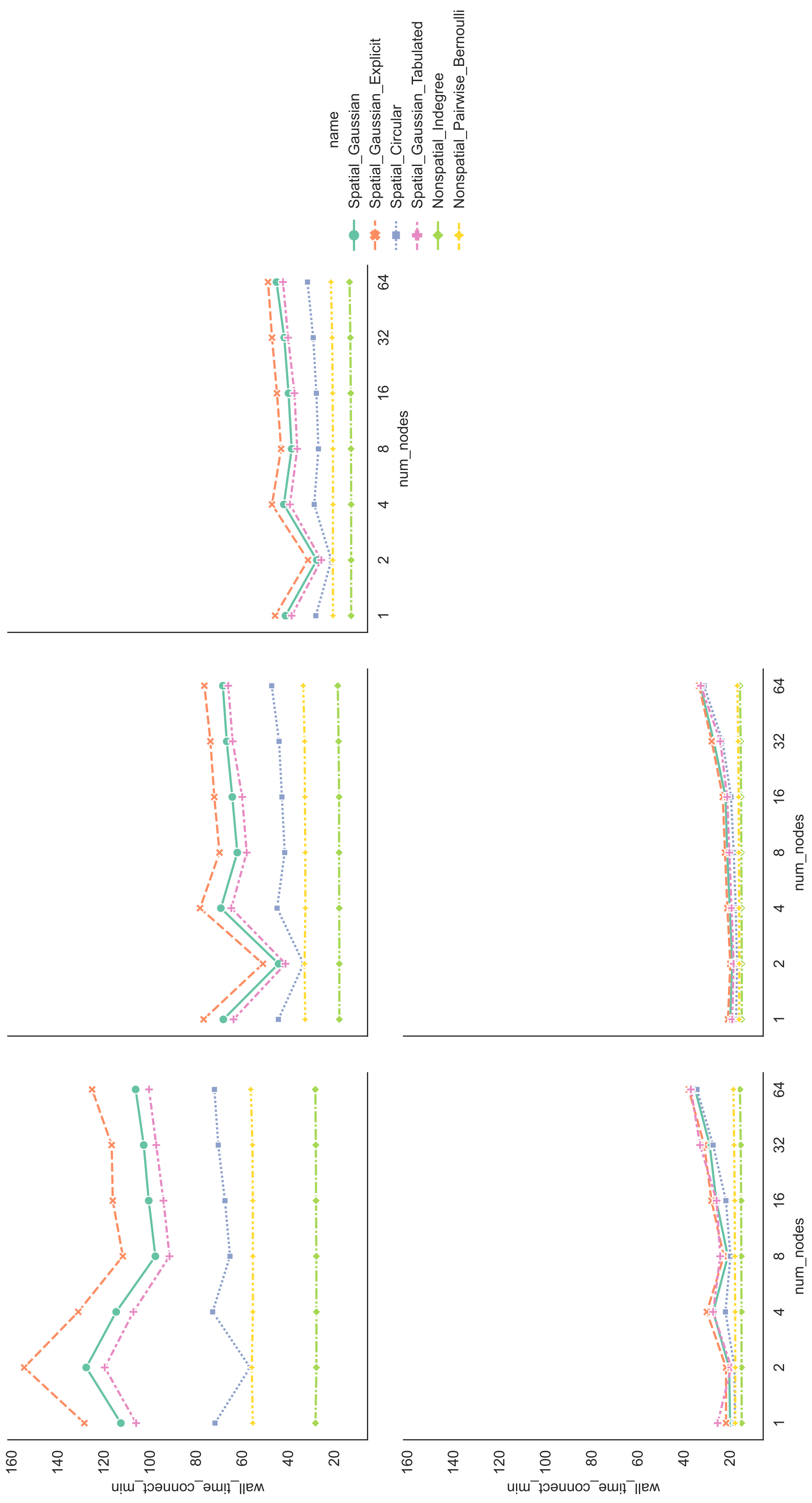

Figure 4.1: Ranking of all network models by the connection time. Starting from top left to bottom right, we have results for 8, 16, 32, 64 and 128 threads per task.

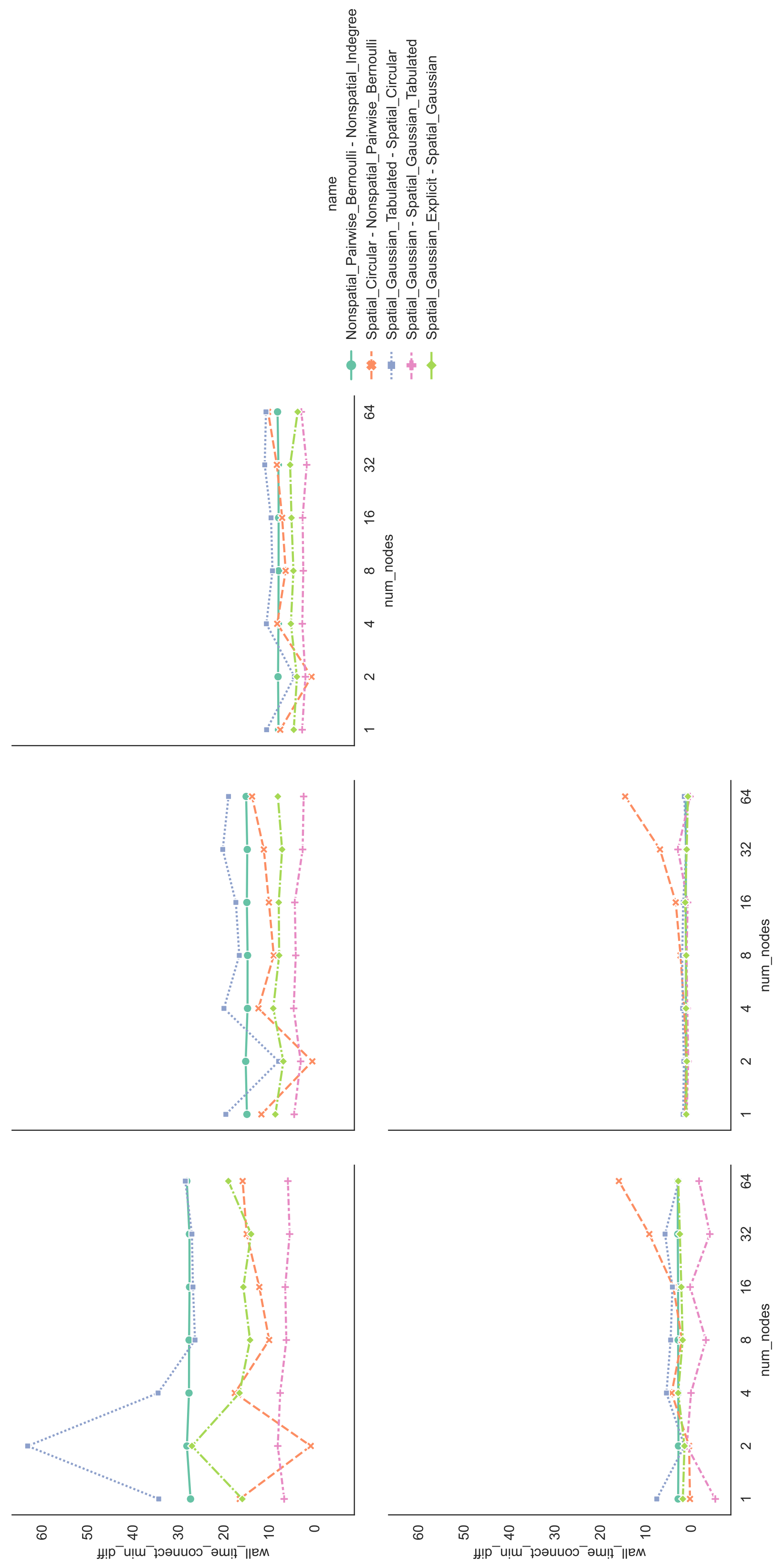

Figure 4.2: Differences in the ranking of all network models. Starting from top left to bottom right, we have results for 8, 16, 32, 64 and 128 threads per task.

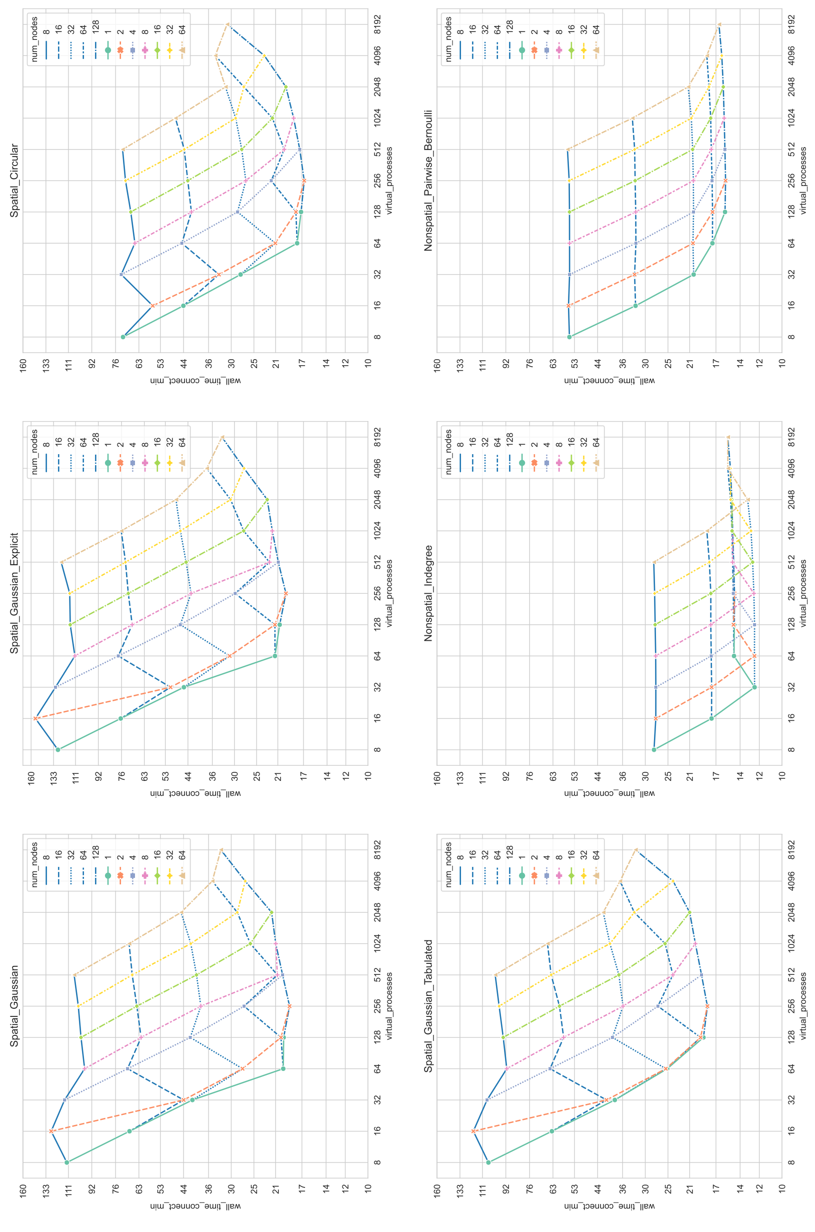

Figure 4.3: Virtual processes.

References

Abi Akar, N., B. Cumming, V. Karakasis, A. Küsters, W. Klijn, A. Peyser, and S. Yates (2019). Arbor—a morphologically-detailed neural network simulation library for contemporary high-performance computing architectures. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 274–282. IEEE.

Albers, J., J. Pronold, A. C. Kurth, S. B. Vennemo, K. H. Mood, A. Patronis, D. Terhorst, J. Jordan, S. Kunkel, T. Tetzlaff, M. Diesmann, and J. Senk (2021). A modular workflow for performance benchmarking of neuronal network simulations https://arxiv.org/abs/2112.09018.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pp. 483–485.

Bauer, P., A. Thorpe, and G. Brunet (2015). The quiet revolution of numerical weather prediction. *Nature 525*(7567), 47–55.

Binzegger, T., R. J. Douglas, and K. A. Martin (2004). A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience 24*(39), 8441–8453.

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of Computational Neuroscience 8*(3), 183–208.

Centre, J. S. (2021). JUBE. https://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/JUBE/_node.html. [Online; accessed 1-January-2021].

Computational and Systems Neuroscience & Theoretical Neuroscience (2021a). beNNch. https://github.com/INM-6/beNNch https://arxiv.org/abs/2112.09018.

Computational and Systems Neuroscience & Theoretical Neuroscience (2021b). Builder. https://github.com/INM-6/Builder https://arxiv.org/abs/2112.09018.

Davison, A., D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger (2008). Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics 2*, 11.

Deepu, R., S. Spreizer, G. Trensch, D. Terhorst, S. B. Vennemo, J. Mitchell, C. Linssen, H. Mørk, A. Morrison, J. M. Eppler, N. L. Kamiji, R. de Schepper, I. Kitayama, A. Kurth, A. Morales-Gregorio, P. Nagendra Babu, and H. E. Plesser (2021, September). NEST 3.1 https://doi.org/10.5281/zenodo.5508805.

Djurfeldt, M. (2012). The connection-set algebra—a novel formalism for the representation of connectivity structure in neuronal network models. *Neuroinformatics 10*(3), 287–304.

Einevoll, G. T., A. Destexhe, M. Diesmann, S. Grün, V. Jirsa, M. de Kamps, M. Migliore, T. V. Ness, H. E. Plesser, and F. Schürmann (2019). The scientific case for brain simulations. *Neuron 102*(4), 735–744.

Eppler, J., M. Helias, E. Muller, M. Diesmann, and M.-O. Gewaltig (2009). PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics 2*, 12, https://doi.org/10.3389/neuro.11.012.2008 https://www.frontiersin.org/article/10.3389/neuro.11.012.2008.

Gewaltig, M.-O. and M. Diesmann (2007). NEST (neural simulation tool). *Scholarpedia 2*(4), 1430.

Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM 31*(5), 532–533.

Gustavsson, A., M. Svensson, F. Jacobi, C. Allgulander, J. Alonso, E. Beghi, R. Dodel, M. Ekman, C. Faravelli, L. Fratiglioni, et al. (2011). Cost of disorders of the brain in Europe 2010. *European Neuropsychopharmacology 21*(10), 718–779.

Halnes, G., S. Augustinaite, P. Heggelund, G. T. Einevoll, and M. Migliore (2011). A multi-compartment model for interneurons in the dorsal lateral geniculate nucleus. *PLoS Computational Biology 7*(9), e1002160.

Hodgkin, A. L. and A. F. Huxley (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology 117*(4), 500–544.

Ippen, T., J. M. Eppler, H. E. Plesser, and M. Diesmann (2017). Constructing neuronal network models in massively parallel environments. *Frontiers in Neuroinformatics 11*, 30, https://doi.org/10.3389/fninf.2017.00030.

Jordan, J., T. Ippen, M. Helias, I. Kitayama, M. Sato, J. Igarashi, M. Diesmann, and S. Kunkel (2018, feb). Extremely scalable spiking neuronal network simulation code: From laptops to exascale computers. *Frontiers in Neuroinformatics 12*, 2, https://doi.org/10.3389/fninf.2018.00002.

Kumbhar, P., M. Hines, J. Fouriaux, A. Ovcharenko, J. King, F. Delalondre, and F. Schürmann (2019). CoreNEURON: an optimized compute engine for the NEURON simulator.

Kunkel, S., T. C. Potjans, J. M. Eppler, H. E. E. Plesser, A. Morrison, and M. Diesmann (2012). Meeting the memory challenges of brain-scale network simulation. *Frontiers in Neuroinformatics 5*, 35.

Kunkel, S., M. Schmidt, J. M. Eppler, H. E. Plesser, G. Masumoto, J. Igarashi, S. Ishii, T. Fukai, A. Morrison, M. Diesmann, et al. (2014). Spiking network simulation code for petascale computers. *Frontiers in Neuroinformatics 8*, 78.

Lapique, L. (1907). Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Pathololgy 9*, 620–635.

Li, X. (2018). Scalability: strong and weak scaling. https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/. [Online; accessed 2-June-2022].

LLC., S. (2019). Slurm Workload Manager . https://slurm.schedmd.com/. [Online; accessed 2-June-2022].

Markram, H. (2013). Seven challenges for neuroscience. *Functional Neurology 28*, 145–151.

McCormick, D. A. and J. R. Huguenard (1992). A model of the electrophysiological properties of thalamocortical relay neurons. *Journal of Neurophysiology 68*(4), 1384–1400.

Message Passing Interface Forum (2021). MPI: A Message-Passing Interface Standard. https://www.mpi-forum.org/docs/. [Online; accessed 2-June-2022].

Migliore, M., E. Cook, D. Jaffe, D. Turner, and D. Johnston (1995). Computer simulations of morphologically reconstructed ca3 hippocampal neurons. *Journal of Neurophysiology 73*(3), 1157–1168.

Morrison, A., C. Mehring, T. Geisel, A. Aertsen, and M. Diesmann (2005). Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural Computation 17*(8), 1776–1801.

National Research Council (1989). *Opportunities in Biology*. Washington, DC: The National Academies Press https://nap.nationalacademies.org/catalog/742/opportunities-in-biology.

Nordlie, E., M.-O. Gewaltig, and H. E. Plesser (2009). Towards reproducible descriptions of neuronal network models. *PLoS Computational biology 5*(8), e1000456.

OpenMP Architecture Review Board (2008). OpenMP Application Program Interface. http://www.openmp.org/mp-documents/spec30.pdf. [Online; accessed 2-June-2022].

Plesser, H. E., J. M. Eppler, A. Morrison, M. Diesmann, and M.-O. Gewaltig (2007). Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In A.-M. Kermarrec, L. Bougé, and T. Priol (Eds.), *European conference on parallel processing*, Berlin, Heidelberg, pp. 672–681. Springer https://doi.org/10.1007/978-3-540-74466-5_71.

Potjans, T. C. and M. Diesmann (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cerebral Cortex 24*(3), 785–806.

RedHat OpenStack Platform (2021). RedHat OpenStack Platform. https://www.redhat.com/de/technologies/linux-platforms/openstack-platform. [Online; accessed 2-June-2022].

Senk, J., B. Kriener, M. Djurfeldt, N. Voges, H.-J. Jiang, L. Schüttler, G. Gramelsberger, M. Diesmann, H. E. Plesser, and S. J. van Albada (2021). Connectivity concepts in neuronal network modeling https://arxiv.org/abs/2110.02883.

Smith, J. M. (1952). The importance of the nervous system in the evolution of animal flight. *Evolution 6*(1), 127–129.

Society for Neuroscience (2018). Brain facts: A primer on the brain and nervous system https://www.brainfacts.org/.

Spreizer, S., J. Mitchell, J. Jordan, W. Wybo, A. Kurth, S. B. Vennemo, J. Pronold, G. Trensch, M. A. Benelhedi, D. Terhorst, J. M. Eppler, H. Mørk, C. Linssen, J. Senk, M. Lober, A. Morrison, S. Graber, S. Kunkel, R. Gutzen, and H. E. Plesser (2022, March). NEST 3.3 https://doi.org/10.5281/zenodo.6368024.

Sterratt, D., B. Graham, A. Gillies, and D. Willshaw (2011). *Principles of computational Modelling in Neuroscience*. Cambridge University Press.

Thomson, A. M., D. C. West, Y. Wang, and A. P. Bannister (2002). Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2–5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro. *Cerebral Cortex 12*(9), 936–953.

United Nations (2001). Mental Health and Development. https://www.un.org/development/desa/disabilities/issues/mental-health-and-development.html. [Online; accessed 11-October-2021].

Van Albada, S. J., M. Helias, and M. Diesmann (2015). Scalability of asynchronous networks is limited by one-to-one mapping between effective connectivity and correlations. *PLoS Computational biology 11*(9), e1004490.

Vieth, B. v. S. (2021). Jusuf: Modular tier-2 supercomputing and cloud infrastructure at jülich supercomputing centre. *Journal of large-scale research facilities JLSRF 7*, 179.

Zaytsev, Y. and A. Morrison (2014). CyNEST: a maintainable Cython-based interface for the NEST simulator. *Frontiers in Neuroinformatics 8*, 23, https://doi.org/10.3389/fninf.2014.00023 https://www.frontiersin.org/article/10.3389/fninf.2014.00023.