I Like To Move It – Computation Instead of Data in the Brain

Fabian Czappa, Marvin Kaster, Felix Wolf

March 2025

The detailed functioning of the human brain is still poorly understood. Brain simulations are a well-established way to complement experimental research, but must contend with the computational demands of the approximately 10^{11} neurons and the 10^{14} synapses connecting them, the network of the latter referred to as the connectome. Studies suggest that changes in the connectome (i.e., the formation and deletion of synapses, also known as structural plasticity) are essential for critical tasks such as memory formation and learning. The connectivity update can be efficiently computed using a Barnes-Hut-inspired approximation that lowers the computational complexity from $O(n^2)$ to $O(n \log n)$, where n is the number of neurons. However, updating synapses, which relies heavily on RMA, and the spike exchange between neurons, which requires all-to-all communication at every time step, still hinder scalability. We present a new algorithm that significantly reduces the communication overhead by moving computation instead of data. This shrinks the time it takes to update connectivity by a factor of six and the time it takes to exchange spikes by more than two orders of magnitude.

1 Introduction

Although directly responsible for our everyday actions, understanding the human brain has remained elusive due to approximately 86 billion neurons, 100 trillion synapses (connections between neurons), and an uncertain number of non-neuronal cells Herculano-Houzel [2009]. Its inner workings and diverse processes are still largely unknown, as is the precise number of neurons Goriely [2024]. Some aspects, however, are known. For example, there is undeniable evidence that synapses change in response to lesions Butz and van Ooyen [2013] or learning Bennett et al. [2018], La Rosa et al. [2020] and in the development phase Fandakova and Hartley [2020]. This fact, along with the ethical concerns and limitations of experiments with human subjects, makes simulations of brain development essential for understanding the larger organizational principles at play.

Which, in turn, is also a challenge: Most models that allow a change in the brain's connectivity (the *structural connectome*) focus on a fixed layout of

synapses and change their weight (called *synaptic plasticity*) Citri and Malenka [2008], Magee and Grienberger [2020]. In contrast, models that describe a change in connectivity are rare Butz et al. [2009]; some focus on the growth of a neuron itself (neglecting the overall picture) Tavosanis [2012], Yamada and Kuba [2016], while others reduce neurons to simple points and focus on the overall dynamic Gallinaro et al. [2022], Kaster et al. [2024]. One specimen of the latter is the model of structural plasticity (MSP) Butz and van Ooyen [2013], which stands alone in its field: It postulates that neurons have a target activity and change their connectivity based on their current and target activity. If their activity falls below their target, they build synapses and thus receive more input, and if their activity exceeds it, they discard synapses and thus receive less input. This homeostatic rule brings the whole brain to an equilibrium where there is little change, and all (most) neurons have an activity close to their respective target. However, this introduces a bottleneck from the computer science perspective, as several aspects of the model have a quadratic time complexity (in the number of neurons): Forming a synapse requires, in principle, calculating a probability from the searching neuron to each potential partner. Previous work uses the Barnes–Hut algorithm Barnes and Hut [1986], Rinke et al. [2018] or the fast multipole method Darve [2000], Nöttgen et al. [2022] to reduce the time complexity of the computational work to quasi-linear Czappa et al. [2023]. Furthermore, whenever a neuron spikes, all connected neurons must be notified. The spike is transmitted via all outgoing synapses. In principle, this has a time complexity of the number of neurons times the number of synapses per neuron; however, in practice, all processes that partake in the simulation have to communicate with each other nonetheless.

In this work, we will develop two algorithms for improving MSP; one building on Barnes–Hut like previous work Rinke et al. [2018], and one approximating the spikes of the neurons. Our work will, just as virtually all previous work, use the Message Passing Interface (MPI) Message Passing Interface Forum [2023] as a convenient standard for processes to communicate, distribute work, and share data. Firstly, we modify the Barnes–Hut approximation to be *location aware* in the following sense: Until now, the neurons have been stored in a spatial octree, which is itself distributed Rinke et al. [2018]. The upper portion is replicated on all MPI ranks, while the subtrees below a certain level are stored at different MPI ranks. Whenever a neuron needs data from an MPI rank other than the one responsible for the neuron, it downloads the data via RMA. In our new version, we send the neuron to the other MPI rank and move the computation instead of the data. Although this does not reduce the computational complexity, it does reduce the communication complexity from logarithmic in the number of neurons to constant on a per-neuron basis.

Secondly, we reduce the bottleneck caused by synchronizing the spiking behavior every time step. Instead, we propose an algorithm to approximate the firing rates of neurons by their frequency. This allows us to drastically reduce the number of synchronization points while preserving the general activity pattern. In contrast to our new connectivity-update algorithm, this one alters the simulation results concerning the underlying model, as it introduces an addi-

tional layer of approximation. While, in theory, this could introduce ripple effects that drastically change the dynamics of the network, in practice, this is not the case. We observe a slight deviation between our approximation and the "ground truth" (also only a model), but this difference has no wider practical consequences.

In Section 2, we review work on neuron models, other simulations of plasticity, full-scale simulations of neuronal activity, and applications of the Barnes–Hut algorithm in computational biology. Following this, in Section 3, we introduce all relevant models for our work as background, including basic terminology. In Section 4, we introduce our two algorithms, explain their pros and cons, and dive into implementation details. Our evaluation in Section 5 focuses on the speed of our algorithms compared to the status quo, the qualitative changes of the simulation results, and a theoretical analysis of our optimizations. Lastly, we conclude our work in Section 6 and give an outlook on what can and should be done (on the computer-science side) in the future of whole-brain structural plasticity.

2 Related Work

There is a plethora of different brain simulators depending on the use case and the research question. NEST Gewaltig and Diesmann [2007] offers a general-purpose simulation framework for spiking neurons; however, in most cases, the connectivity of the neurons remains fixed. While allowing manual edits to the connectivity, it also implements MSP Butz and van Ooyen [2013] with a time complexity of $O(n^2)$ with n being the number of neurons (i.e., without the Barnes–Hut approximation), which limits scaling. Other brain simulators, such as Arbor Abi Akar et al. [2019], allow changes in connectivity and model the neurons with far more detail. For Arbor, specifically, one must write the code to change the plasticity oneself. So every model—in theory—is possible to implement. NEURON Carnevale and Hines [2006] focuses on morphologically detailed models, including multi-compartment models, ion channels, and cable equations. Brian2 Stimberg et al. [2019], on the other hand, is designed for simple neuron morphology but enables users to write the differential equations themselves, making it very flexible and easily extendable. CARLSim Niedermeier et al. [2022] is a simulator designed to handle large-scale simulations on heterogeneous clusters supporting CPU and GPU architectures. Of all these simulators, only NEST supports structural plasticity directly. The other simulators allow the users to change the connectivity, but the user must write the code to determine when and where synapses must be built or removed. There are many other simulators available that do not simulate single neurons, such as the Virtual Brain Sanz Leon et al. [2013] that aims for a simulation of the full-brain dynamics at a much lower resolution (and without changes in connectivity) or that do not model single spikes, such as Nengo Bekolay et al. [2014].

Other large-scale simulations include a simulation of 10^9 neurons with 10^{13} synapses in 2009 Ananthanarayanan et al. [2009], $1.86{\cdot}10^9$ neurons and $11.1{\cdot}10^{12}$

synapses with NEST in 2014 Kunkel et al. [2014], and $68 \cdot 10^9$ neurons $5.4 \cdot 10^{13}$ synapses Yamaura et al. [2020]. However, all these large-scale simulations have a fixed connectivity of neurons, removing the largest burden from the simulation—a flexible communication structure for the spikes and a general way to connect neurons arbitrarily during the simulation.

MSP has the structure of an *n-body problem*: Given n bodies, calculate how they interact. This structure is typical in physics Pfalzner and Gibbon [1996], where the interacting bodies range from elementary particles to solar systems and beyond Costa et al. [2005]. Usually, the solution has a computational time complexity of $O(n^2)$, as every body interacts with every other body. Naturally, many different avenues of approximation have already been explored. For example, the bodies that act can be grouped if they are far enough away—at large distances, it does not matter if a body is slightly mispositioned or if a single force (of double the potential) is used instead of two separate forces. This idea underlies the Barnes–Hut algorithm Barnes and Hut [1986], Barnes [1990], Burtscher and Pingali [2011], which groups bodies together and builds a spatial octree (the root representing the whole domain, its eight children each octant, ...). The algorithm employs an approximation parameter θ (usually within the range $[0.1, 0.4]$) and allows the approximation of the received forces from a subdomain if the subdomain is small enough for the distance to its target (length of the subdomain divided by the distance to the center of mass must be smaller than *theta*). The time complexity for the Barnes–Hut algorithm with $\theta > 0$ is $O(n \cdot \log n)$, and with $\theta = 0$, the algorithm computes the direct solution. Grouping the bodies that are acted upon as well results in the fast multipole method (FMM) Darve [2000], Rokhlin [1985], which has a time complexity of $O(n)$. Furthermore, shifting the view from an algorithmic standpoint to an absolute one, the underlying motive is to (implicitly) compute the $n \times n$ matrix of interactions. If this matrix is of low rank, methods from numeric linear algebra can be used to calculate it efficiently Ceruti et al. [2024], although this is not the case for our problem.

MSP, however, has a twist that problems in physics can avoid: While for a star, the gravitational force a distant galaxy exercises is enough, MSP must know more. It is not enough to know that a new synapse reaches into a brain region, the new partner must be an actual neuron. So, previous work has adapted the Barnes–Hut algorithm Rinke et al. [2018], recursively applying it until an actual neuron is found (which does not change the time complexity of $O(n \cdot \log n)$ Czappa et al. [2023]). FMM can also be adapted Nöttgen et al. [2022]; yet, because of the disadvantage of having to find an actual partner, the adapted version also has a time complexity of $O(n \cdot \log n)$. Furthermore, as a neuron needs an actual target, FMM restricts the choice of closely aligned neurons that search for a partner at the same time—they have to connect to the same region.

The Barnes–Hut algorithm is used in many aspects of computational biology besides modeling the structural plasticity in the brain. For example, early works use it for approximating molecular dynamics Plimpton [1995]. It can also be used outside of the direct problem domain, e.g., for speeding up a t-SNE clas-

sification Rukhsar and Tiwari [2023] or a visualization Hadjiabadi et al. [2021] with the aim of helping in seizure or epilepsy models.

3 Background

We recapitulate definitions, observations, and algorithms from publications such as the one introducing the MSP Butz and van Ooyen [2013] and its approximation Rinke et al. [2018]. We will focus on the source of parallelism when using the adapted Barnes–Hut algorithm, and review how a spatial octree is used to speed up the calculation.

3.1 The model of structural plasticity

The model does what its name promises: It models a way of computing structural plasticity: how neurons form new synapses and delete existing synapses over a fixed period. It consists of three distinct internal phases that repeat until completion:

1. The update of electrical activity.
2. The update of synaptic elements.
3. The update of connectivity.

The update of the synaptic elements affects both a neuron's axon (its part that sends out electrical signals to other neurons) and its dendrite (i.e., the part that receives signals from other neurons). However, the update of synapses only happens every 100th time step as a measure against high fluctuations.

Update of electrical activity First, the neurons exchange their spikes from the last update step (computed, e.g., with the Izhikevich model Izhikevich [2003]), that is, the spikes travel across the synapses to the neurons they are connected to via their axons. Then, each neuron uses its electrical state and the input it receives (via its dendrite or as background noise) to calculate its next electrical state. If the neuron model determines that a neuron spiked, this spike is saved for the next time step, during which this spike is transferred via the axon. Lastly, the neurons update their intercellular calcium level—a floating average of the firing rate that acts as a dampening layer.

Update of synaptic elements Each neuron updates its axon's and dendrite's grown synaptic elements. This update is based on the current and target calcium levels. A lower-than-target level induces the growth of synaptic elements, while a higher-than-target level retracts synaptic elements.

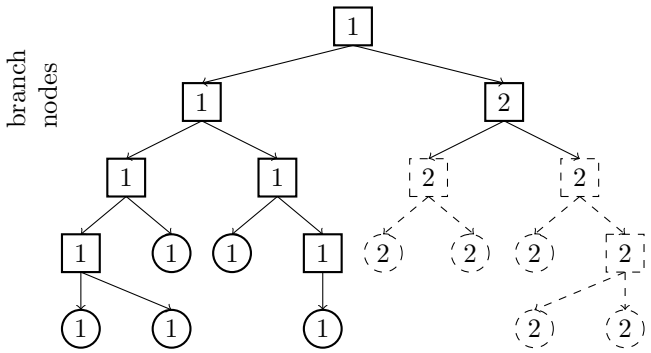

Figure 1: Example of a distributed tree (shown is a binary tree for the sake of simplicity) from the view of the first process. Boxes are inner nodes, and circles are leaf nodes; the numeric label indicates the process of owning the node. As we depict the view from process 1, it owns the (replica of the) root node. Solid lines indicate information on process 1, and dashed lines indicate information that resides on other ranks.

Update of synapses This phase consists of two sub-phases. Firstly, if a neuron retracts a vacant synaptic element, it is removed without affecting any synapse. However, suppose the neuron retracts a synaptic element while all are bound in synapses. In that case, one is chosen randomly, the synapse is broken, and the affected partner has a new vacant synaptic element (and one synapse less). Secondly, all neurons with a vacant axonal element search for a matching vacant dendritic element on another neuron. This is done by evaluating a probability kernel based on the distance between the searching neuron and potential partners; one potential partner is picked randomly according to the probability distribution, and a synapse is proposed. Then, once all synapses are proposed, the neurons accept as many requests for new synapses as possible (randomly) based on their vacant dendritic elements and the number of received requests. Requests that cannot be fulfilled because a neuron received too many requests for its number of vacant dendritic elements are declined. The axonal elements receive the response (succeeded/failed) and are bound in a synapse or remain vacant for the next update of synapses.

3.2 The adapted Barnes–Hut

The most time-consuming part of MSP is the update of synapses, even though it is only executed every 100 steps. As every vacant axonal elements has to calculate its probability of forming a connection to every vacant dendritic element, this can have a quadratic time complexity in the number of neurons. To alleviate this, previous work adapted the Barnes–Hut Rinke et al. [2018] algorithm or

the fast multipole method Nöttgen et al. [2022] to reduce the time complexity of the computation to quasi-linear Czappa et al. [2023]. This is done by (a) a division of the simulation domain, (b) an induced spatial octree, and (c) an induced division of work.

Division of the simulation domain Given a simulation domain $D = [0, l_x] \times [0, l_y] \times [0, l_z] \subseteq \mathbb{R}^3$ and a number of MPI ranks k (a power of two), find the smallest b such that $8^{b-1} \leq k < 8^b$, and divide D into 8^b subdomains, each of size $(l_x/b) \times (l_y/b) \times (l_z/bp)$, which then cover D. These subdomains are indexed by a space-filling curve (e.g., the Morton curve), and each MPI rank is responsible for 1, 2, or 4 consecutive subdomains (in the sense of the space-filling curve).

Spatial octree The MPI ranks collectively construct a spatial octree for all neurons in the simulation domain. All share a common upper part: The root node represents the whole simulation domain; its eight children represent the eight octants one receives by halving the simulation domain along the x-, y-, and z-axis, ..., repeated until the level b (determined in the previous paragraph). On level b reside the *branch nodes*, each representing one of the 8^b subdomains. Going further down in the tree, starting at level $b + 1$, only the MPI rank that owns this subdomain has actual data. It subdivides its subdomain until each cell (=leaf node in the octree) contains at most one neuron. Figure 1 shows such a distributed tree.

Division of work Before the update of synapses, the MPI ranks update the distributed octree with respect to the vacant synaptic elements. The leaf nodes not containing an actual neuron have no valid position and no vacant dendritic elements; the ones containing a neuron have the neuron's position as their own and as many vacant dendritic elements as the neuron. The inner nodes have as many vacant dendritic elements as all of their children combined, and their position is the weighted average of their children. The MPI ranks update their octree bottom-up up to the branch nodes, where they all-to-all exchange their branch nodes and then update further up to the root node. This way, each MPI rank knows how many vacant synaptic elements are up to the level of branch nodes and where approximately those are (as in the average position).

If a neuron has a vacant axonal element, thus wanting to form a synapse, the MPI rank in whose subdomain the neuron resides is responsible for all resulting calculations. It starts with the root node as a potential target, evaluates the acceptance criterion of the Barnes–Hut algorithm (cell-length divided by distance smaller than θ; will always reject the root node), and recursively substitutes a node's eight children whenever the node is rejected. One is randomly chosen based on the connection probability from a list of nodes that satisfy the acceptance criterion. All is well if the target is a leaf node and a synapse-formation request is formed. This request contains:

- The ID of the source neuron

- The ID of the target neuron
- The cell type of the target node; i.e., excitatory or inhibitory

However, if the target node is an inner node, the whole procedure has to start again with the target node instead of the root node. Whenever the MPI rank requires octree nodes that it does not own, it downloads those via RMA and attaches them to its own octree; they stay valid until the end of the whole synapse-formation phase and thus do not have to be re-downloaded for the next neuron that might need them.

Once all formation requests have been formed, these are all-to-all submitted, locally accepted/declined, and their responses are returned. Note here that a simple *yes*/*no* is sufficient as an answer, as the requesting neuron knows which partner it has chosen.

4 Communication-Efficient Algorithms

In this section, we explain our rationale behind both proposed algorithms for updating connectivity and exchanging neuron spikes. We will show our reduction in (theoretical) communication complexity and (practical) communication implementation.

4.1 Location-aware Barnes–Hut

The main bottleneck when it comes to the division of work (3.2) is not the actual computations but the waiting and transfer times of the data. Although using RMA removes the need for the "sender" to be aware of the communication, the data transfer must still be performed. Furthermore, the structure of the algorithm does not lend itself to hiding the waiting times with other computations: When requesting *far* nodes, the computation has already reached a point where it can only proceed once the data has arrived.

In our work, we propose a hybrid approach that removes quite a lot of communication: The first step remains the same as in 3.2: A neuron with a vacant axonal bouton searches for a partner starting at the root node. If the returned node is above or below the level of the branch nodes, this is repeated until a target node that resides at or below the branch level is found. If, during these searches, nodes stored on other MPI ranks are required, we download them with MPI just as before. However, if the returned node is at or below the level of branch nodes, we instead send a request for *synapse formation and calculation* to the MPI rank responsible for the node. This request contains:

- The ID of the source neuron
- The position of the source neuron
- The ID of the target node
- The type of target node; i.e., if the target is already a leaf

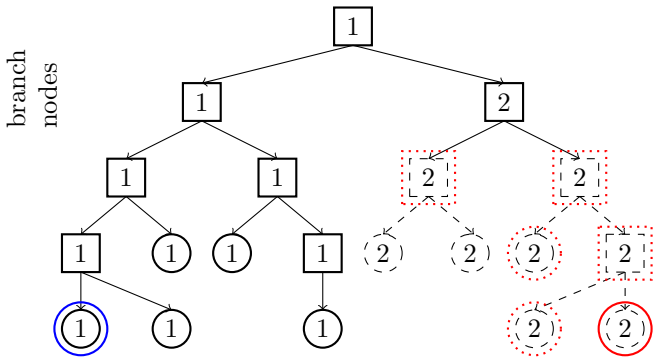

Figure 2: The neuron from process 1 (the leaf node with the blue solid marking) will propose a synapse to the neuron from process 2 (the leaf node with the red solid marking). In the default Barnes–Hut implementation, process 1 must download all nodes marked red via remote-memory access. In our proposed algorithm, process 1 sends only a part of the information from the neuron to process 2. Thus, the transfer direction has mostly been reversed, and the computation has been moved to the target node.

- The cell type of the target node; i.e., excitatory or inhibitory

All MPI ranks construct such requests and all-to-all exchange them. In the implementation, an old request has a size of `8 + 8 + 1 = 17` B, while the new request has a size of `8 + 24 + 8 + 1 + 1 = 42` B.

Having received such requests, the MPI ranks check the target node type. If it is a leaf, they do not have to compute anything and can convert the request to the old form (source ID, target ID, type). Otherwise, the target node is an inner node of the octree (note that we are on the MPI rank that owns the respective part of the tree). The MPI rank that owns the target node starts searching at the specified node with the node's position that wants to form a synapse (i.e., from the other MPI rank). This search requires no further RMA communication but a more lengthy response: Instead of *yes*/*no* (`1` B), the answer now contains the ID of the found neuron (if any) and an indicator whether the search was successful (just as before, too many requests can reach a neuron and thus, it has to decline some)—with a size of `8 + 1 = 9` B. Figure 2 shows a schematic representation of the communication requirements of the old algorithm versus the new algorithm.

4.2 Approximation of firing rates

Besides the formation of synapses, the second bottleneck in the simulation is the exchange of spikes. Whenever a neuron spikes, this action potential travels through its axon across a synapse to the dendrites of other neurons, which

then receive the spike as input in the next simulation step. This is—strictly speaking—not an all-to-all communication, as neurons have a fixed number of partners no matter how many neurons take part in the simulation, yet in practice, this becomes an all-to-all communication between the MPI ranks involved because a neuron usually maintains connections to so many other neurons (around a thousand) that it needs to talk to the majority of MPI ranks.

Any standard neuron model is *time-critical* concerning the received spiking times, i.e., the state transitions $s_0 \xrightarrow[\text{spike}]{\text{receives}} s_1 \xrightarrow[\text{no spike}]{\text{receives}} s_2$ and $s_0 \xrightarrow[\text{no spike}]{\text{receives}} s_1' \xrightarrow[\text{spike}]{\text{receives}} s_2'$ result in different ending states. Only for the most fundamental state transitions would $s_2 = s_2'$ hold (e.g., simple combinations of minimum, sum, maximum, ...), although even in such cases, s_1 and s_1' need not be equal. In general, state transitions with different numbers of spikes result in different ending states as well.

However, what leads us to our proposed algorithm is that the previous model for neuronal spikes already approximates various aspects. In reality, spikes are not transmitted instantaneously; they occur over a prolonged period, and their effect depends on the distance the spike travels, the exact contact position on the receiving neurons, and more. With this in mind, we propose to approximate the spikes generated by the neuron model by a firing frequency—instead of the individual spikes, the frequency is transmitted every so often, and the receiving neurons use a pseudo-random number generator and this frequency to determine if the sending neuron spiked. We only use this model for transmitting spikes between MPI ranks, as for a connected neuron pair on the same MPI rank, checking whether one spiked is virtually free.

This algorithm has the benefit of significantly reducing the number of synchronization points. Instead of synchronizing and exchanging data in every time step, we define an epoch length Δ and exchange the firing frequencies of the neurons only for every Δ time step. This comes with a few drawbacks; none, however, have substantial implications for the simulation. Firstly, exchanging frequencies might result in a more significant data transfer than even Δ transmissions of spikes (due to a different data type and the necessity to send data between each connected neuron pair instead of only those where the sending neuron spiked). Secondly, because neurons do not have a steady firing frequency during the development phase, our model introduces a response lag. Lastly, the spikes of a neuron are no longer synchronized. For example, given a neuron that spikes every ten time steps and three connected neurons, one on the same MPI rank and two on different MPI ranks, it is only given that the first one receives the actual spikes and the other two determine at each time step with a probability of ten percent whether the neuron spiked. In total, the received spikes need not correlate in theory and could have an arbitrarily large mismatch—although they do in practice.

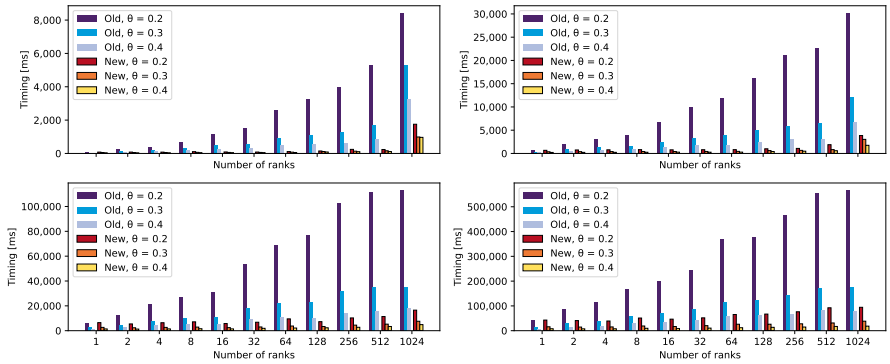

Figure 3: Timing plots for the *old* Barnes–Hut algorithm and our proposed *new* Barnes–Hut algorithm, given for varying numbers of neurons per rank: 1024 neurons per MPI rank on the top left, 4096 neurons per rank on the top right, 16 384 neurons per rank on the bottom left, and 65 536 neurons per rank on the bottom right.

5 Results

We will begin our results with theoretical observations as to where we changed the complexities of the algorithms. Then, we will check our timings on actual hardware, and lastly, we will briefly investigate the amount of transferred data.

5.1 Theoretical benefits

To discuss the theoretical benefits, we will fix a scenario that we can use throughout the analysis. Firstly, the root node of the spatial octree is named R and resides on the level 1 of the tree. Its children are on level 2, its grandchildren on level 3, and so on. The branching level is b like before. We fix two neurons, N and N', on levels l and l', respectively. If the tree is balanced, $l = l' \in O(\log n)$ holds. With each of the two neurons, we associate a path $(P = (R, ..., N), P' = (R, ..., N'))$ from the root node down the tree. The neuron N will belong to MPI rank R. Lastly, neuron N will fire every K simulation steps, while neuron N' will fire every K' simulation steps.

Update of connectivity We start with the differences in the connectivity update, i.e., the adapted Barnes–Hut Rinke et al. [2018] and our location-aware version. For this, we must distinguish two scenarios: Firstly, assume N proposes to a local neuron, i.e., N' also belongs to MPI rank R. For this to happen, all nodes on P' belong to R as well. No matter if N chose N' in the first application of the Barnes–Hut algorithm or it chose some inner nodes from P' first and then, in a repeated application, chose N' later; it never started a search on a node

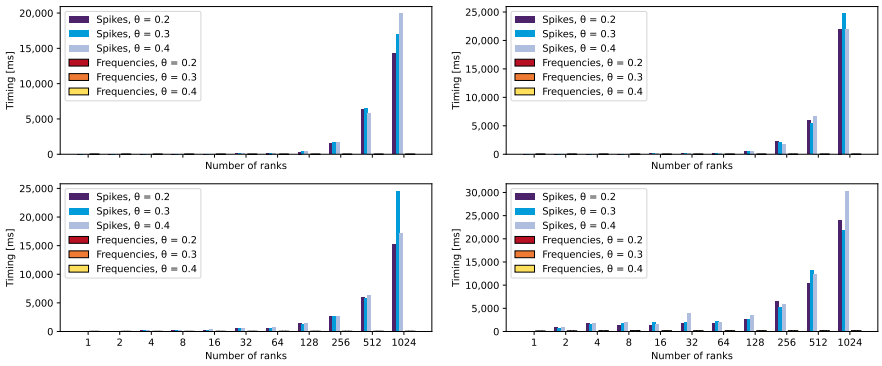

Figure 4: Timing plots for the transfer of neuron *spikes* and neuron firing *frequencies*, given for varying numbers of neurons per MPI rank: 1024 neurons per rank on the top left, 4096 neurons per rank on the top right, 16 384 neurons per rank on the bottom left, and 65 536 neurons per rank on the bottom right. Note that transferring the frequencies is virtually free.

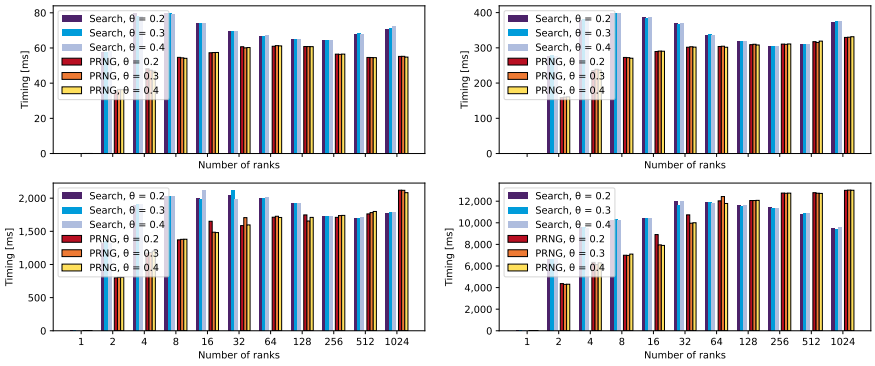

Figure 5: Timing plots for look-up of spikes from distant neurons (by a binary *search*) and for the approximation of spikes based on the frequency with a pseudo-random number generator (*PRNG*), given for varying numbers of neurons per MPI rank: 1024 neurons per rank on the top left, 4096 neurons per rank on the top right, 16 384 neurons per rank on the bottom left, and 65 536 neurons per rank on the bottom right.

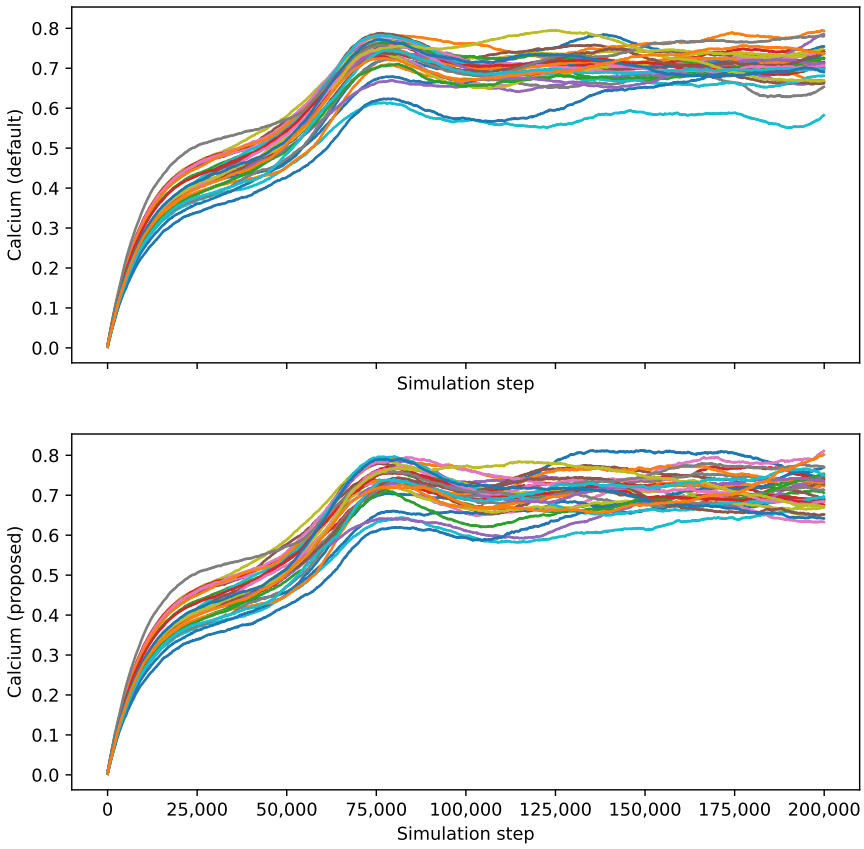

Figure 6: Calcium concentration for the 32 neurons during a simulation of 200,000 simulation steps (2000 connectivity updates). Shown on top is the default way of sending the neuron spikes; shown on the bottom is the proposed way of sending the firing frequencies.

belonging to a different MPI rank, so both the original version and our version perform in the same way.

Secondly, assume that N proposes to a distant neuron, i.e., N' belongs to an MPI rank $R' \neq R$. For this to happen, on the path P', the first nodes up to the branching level b belong to R, while all below belong to R'. While the node at the branching level also belongs to R', R has a local copy. If N had not picked any inner nodes of P' (again, because the acceptance criterion never allowed an approximation on any level), both versions perform the same. Any inner node picked above the branching level is irrelevant, as every MPI rank has a copy of the upper part—no communication involved. In the remaining case, where N picked inner nodes on the path from the root to N' at or below the branching level, our algorithms differ: The original version would download all necessary information via RMA. In contrast, our version sends the information of N to R' and receives a response.

So, what does this entail? The length of P' is level(N'), with b nodes held locally on R and level(N') $- b$ held on R'. In the worst case, this means level(N') $- b - 1 \in O(\log n)$ searches starting at a foreign node—with the required RMA communication—for the original version. Our version sends the information for N to R' as soon as an inner node from P' belonging to R' is picked. The computation of R' with N is analogous to the computation R would perform, albeit in a different state in the pseudo-random number generator.

In conclusion, we reduce the per-neuron communication of $O(\log n)$ to $O(1)$ in the worst case. In theory, a rigorous statistical proof of the number of times a local/distant neuron is chosen with zero/one/two/... intermediate inner nodes is possible, but beyond our capacity and, as Section 5.2 will show, not be necessary in practice.

Update of electrical activity We compare the direct communication of spikes in the original version and the approximated firing frequencies. We assume that we update the frequencies at which neurons fire every Δ simulation steps. The analysis is comparably simple: Firstly, we reduce the number of communications by the factor Δ. Note that in the original version, each rank—even if it will not receive any incoming spikes—has to obtain the number of incoming spikes (i.e., 0 in this case). Secondly, looking at a period of Δ updates and assume the synapse from N to N', the overall communication in the original version is in $O(K * 64\,\text{Byte})$ (it sends the neurons' IDs when they fired). Furthermore, when determining if neuron N spiked, R' will search for the ID of N in all the IDs it received from R. These are sorted, so this is a binary search. In our proposed version, the overall communication is in $O(64\,\text{Bytes})$ and instead of the lookup, the rank draws a pseudo-random number.

Thus, our version can have a theoretical benefit for increasing values of Δ (i.e., reducing the frequency updates the longer the simulation runs and stabilizes). Typical neurons fire with a frequency between 10 Hz and 100 Hz, corresponding to $K, K' \in [10, 100]$, as one simulation step corresponds to 1 ms of biological time. We will choose $\Delta = 100$ (i.e., update the frequency every time

the connectivity changes), and the timings in Section 5.2 show that—although this is quite often—we have an enormous edge; even without a theoretical benefit.

5.2 Timings

We have performed our experiments on the *anonymous high-performance cluster. This paragraph is left out because of the double-blind review.*

Our experimental setup was constant across all experiments: We simulated 1001 update steps of the network (i.e., 10 updates of plasticity), with no initial connectivity, and each neuron having between 1.1 and 1.5 vacant synaptic elements. This ensures that each neuron can connect to one other neuron, and in turn can be connected to by one other neuron. We use 10 updates of plasticity because multiple neurons can choose the same target (which can only accept one), and thus have to retry in the next update.

We ran our tests with {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024} MPI ranks, having {1024, 4096, 16 384, 65 536} neurons each, with an Barnes–Hut acceptance criterion of {0.2, 0.3, 0.4} (so $11 \cdot 4 \cdot 3 = 132$ configurations); and each configuration once with the old algorithms and once with our improved algorithms. We report the average times it took to exchange the fired status and the time it took to form new synapses (which includes finding the targets, exchanging the requests, handling the requests, and exchanging the answers). Furthermore, we report the approximate number of bytes sent, received, and remotely accessed by the MPI ranks. Here, "approximate" means that we only count the bytes we are responsible for, not what the library additionally communicates.

Update of connectivity We give the timings for both Barnes–Hut algorithms—the default one and our proposed one—in Figure 3. It is evident that with a larger θ (allowing more approximations), the connectivity update consumes less time. This is to be expected. Note that small values for the number of MPI ranks or the number of neurons per MPI rank inhibit what we identify as noise-related issues. The timings for one MPI rank are essentially the same—as expected; our algorithm improves the communication between MPI ranks. The timings of our algorithm for "small" simulations (approximately until 64 MPI ranks with 4096 neurons each) show no scaling as the general synchronization for the all-to-all communication and the actual computation dwarfs the required communication. Besides this artifact, the larger the number of MPI ranks, the bigger the benefit of our new algorithm, up to a factor of 6 improvement for the largest simulation (or a factor of 10 improvement for 512 MPI ranks, with 16 384 neurons per MPI rank). Our new algorithm also shows the same scaling behavior as the old one with regard to θ, and the overall scaling behavior is similar to the original algorithm—just vastly faster.

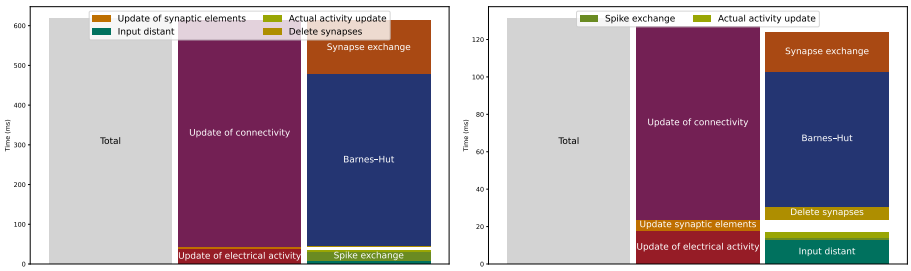

Figure 7: Timings of the simulation with 1024 MPI ranks, 65,536 neurons per rank, and $\theta = 0.2$. Note that for the timings for the old algorithm, "Barnes–Hut" includes remote memory accesses. Left: old algorithm. Right: proposed location-aware algorithm.

Update of electrical activity Figure 4 shows the times for both the default version of transferring neuron spikes and the proposed version of transferring neuron frequencies instead. For the old algorithm, we see slightly more than a doubling of the time it takes to exchange the firing IDs when doubling the number of MPI ranks, and scaling that starts as higher-than-quadrupling for a small number of MPI ranks but drops to lower-than-doubling for a large number of MPI ranks when quadrupling the number of neurons per MPI rank. This indicates that we are on the right track—and the synchronization and the establishment of the communication channels are the main bottleneck. The timings for looking up the spikes once they have been received are essentially constant regarding the number of MPI ranks—albeit an increase from 1 to 2 to 4 ranks (see Figure 5). When it comes to our proposed algorithm, we see a clear trend: The communication effort is virtually constant regarding the number of MPI ranks, only scaling approximately with the number of neurons per rank. However, the times are many orders of magnitude lower than the default version (in the largest simulation: 23 s versus 169 ms).

Looking up the frequencies of the neurons and drawing pseudo-random numbers takes more time than searching the IDs, but not significantly more (9467 ms versus 13 s), see Figure 5. From 64 MPI ranks upwards, the look-up times stay essentially constant. Overall, this slowdown is a price worth paying for the speedup we see in Figure 4.

5.3 Amount of communication

Tables 1 and 2 show the number of bytes transferred during the simulations. For the old Barnes–Hut algorithm and the standard way of transferring the neuron spikes, we list the number of bytes sent and remotely accessed. For the proposed location-aware Barnes–Hut algorithm and the approximation of neuron spikes, we only list the number of bytes sent, as there are no remotely-accessed bytes. In principle, there could be remotely-accessed bytes, but this case does not occur in practice as no MPI rank needs to fetch data. In the small simulations,

our new algorithms transfer more bytes but are quickly overtaken. As a direct consequence, we also reduce the memory footprint "en passant": Where MPI ranks in the old algorithm needed to download octree nodes that belong to another MPI rank and cache them locally in case a different neuron needed the data during the same connectivity update, this is now virtually obsolete.

5.4 Quality of the approximation

For this test, we change the simulation setup significantly. Firstly, we simulate only 32 neurons, distributed to 32 MPI ranks (i.e., every rank is responsible for one neuron)—this forces the neurons to only connect to neurons belonging to other MPI ranks, thus fully using our approximation of spikes. Secondly, we simulate 200,000 steps, having 2000 updates in connectivity. We set the target activity of the neurons to 0.7 and the growth rates of the synaptic elements to 0.001. This results in the neurons wanting 22–23 synapses. Lastly, we also give the neurons a background activity from $\mathcal{N}(5, 1)$ to stimulate them enough so they grow synaptic elements (see Butz and van Ooyen [2013] for a deeper discussion of this choice; same background activity in both simulations).

Figure 6 shows the calcium concentrations for both the standard way of transmitting neuron spikes and our proposed way of approximating the spikes via a firing frequency. The initial phase is similar for both simulations: After the background activity lifts the neurons to approximately 0.4 calcium, they start growing synapses. They overshoot their target activity, prune some synapses, and then try to achieve the target activity. We see the same fluctuations in both simulations. Using the default way of transferring the neuron spikes leaves one neuron out—just because the other neurons have largely connected among themselves and there is no vacant axon of another neuron to connect to its dendrite. In a larger simulation, this is very unlikely to happen, and thus poses no concern to us.

5.5 Total time

The timings we receive for the longest simulation we performed—1024 MPI ranks, 65,536 neurons per MPI rank, and $\theta = 0.2$—are shown in Figure 7. The old algorithms for forming synapses and exchanging neuron spikes, together with the computation of the model for electrical activity, the model for synaptic elements, and model for synapse deletion, took 617 s for 1001 steps. The new algorithms for forming synapses and exchanging neuron spikes, together with the models we did not change, took 131 s for 1001 steps. This is an absolute reduction in wall-clock time of 486 s, which corresponds to a relative reduction of approximately 78.8 %. Looking at the time distribution of the simulation with our new algorithms also points to the potential problems for even larger simulations: With our proposed algorithm, 22 s of the 131 s (roughly 16 %) are easy per-neuron calculations ("Input distant", "Actual activity update", "Update of synaptic elements"), i.e., numerically solving differential equations or

MPI ranks	Neurons per rank 1024	4096	16 384	65 536
1 r.	86 KB	324 KB	1273 KB	5075 KB
	0 B	0 B	0 B	0 B
2 r.	845 KB	3180 KB	12 MB	48 MB
	3111 KB	11 MB	21 MB	152 MB
4 r.	2442 KB	8921 KB	34 MB	136 MB
	8878 KB	53 MB	189 MB	392 MB
8 r.	5956 KB	20 MB	78 MB	312 MB
	55 MB	112 MB	575 MB	2087 MB
16 r.	15 MB	46 MB	170 MB	666 MB
	166 MB	510 MB	1107 MB	5576 MB
32 r.	39 MB	102 MB	358 MB	1379 MB
	331 MB	1311 MB	4141 MB	9 GB
64 r.	108 MB	237 MB	755 MB	2826 MB
	1143 MB	2362 MB	9548 MB	31 GB
128 r.	437 MB	698 MB	1741 MB	5912 MB
	2739 MB	7495 MB	16 GB	72 GB
256 r.	1315 MB	1862 MB	3955 MB	12 GB
	5177 MB	16 GB	49 GB	128 GB
512 r.	3966 MB	5577 MB	9775 MB	25 GB
	14 GB	29 GB	106 GB	357 GB
1024 r.	17 GB	24 GB	32 GB	65 GB
	29 GB	77 GB	188 GB	772 GB

Table 1: Bytes sent and remotely-accessed for the old Barnes–Hut algorithm, the standard way to exchange the neuron spikes, and a small amount of bookkeeping. The values do not significantly differ when varying θ (in the one-digit percent range), so we provide the values for $\theta = 0.2$. The upper values represent the total number of bytes sent/received, the lower values represent the total number of bytes remotely accessed. Digits after the decimal point are cut. 1 KB = 1024 B, etc.

MPI ranks	Neurons per rank 1024	4096	16 384	65 536
1 r.	211 KB	814 KB	3230 KB	12 MB
2 r.	757 KB	2863 KB	11 MB	44 MB
4 r.	1794 KB	6914 KB	26 MB	107 MB
8 r.	3925 KB	14 MB	58 MB	233 MB
16 r.	9908 KB	32 MB	123 MB	486 MB
32 r.	19 MB	66 MB	251 MB	993 MB
64 r.	41 MB	134 MB	509 MB	2006 MB
128 r.	192 MB	380 MB	1132 MB	4140 MB
256 r.	401 MB	778 MB	2286 MB	8318 MB
512 r.	867 MB	1628 MB	4648 MB	16 GB
1024 r.	8685 MB	10 GB	15 GB	39 GB

Table 2: Bytes sent for the proposed location-aware Barnes–Hut algorithm, the proposed way to exchange the neuron frequencies, and a small amount of bookkeeping. The values do not significantly differ when varying θ (in the one-digit percent range), so we provide the values for $\theta = 0.2$. Digits after the decimal point are cut. 1 KB = 1024 B, etc.

drawing pseudo-random numbers. 28 s (roughly 21 %) are spent during communication periods ("Synapse exchange", "Spike exchange", "Delete synapses") (NB: there were no synapses deleted; the timing is just synchronization). All the while, we spent 72 s (roughly 55 %) in the Barnes–Hut computation, which does not include remote memory accesses. This part also occurs on a per-neuron basis, i.e., does not require interactions with other neurons. But, for different neurons, the repeated Barnes–Hut computations have vastly different execution paths (as two neighboring neurons can choose targets in opposite parts of the brain). This contrasts with the earlier-mentioned per-neuron work, which is the same for every neuron.

6 Conclusion

We have presented two algorithms for improving the runtime of MSP-based simulations, which can be used to forecast changes in the (human) brain after learning, lesions, or ordinary development. Building on the Barnes–Hut algorithm and an older algorithm from 2018 Rinke et al. [2018], we improve the time for a connectivity update by a factor of six, the time for exchanging the neurons' spikes by a factor of more than 100, and the amount of transferred information by a factor of 21. We pay with an increase in looking up neuron spikes by a factor of roughly 1.5, an insignificant effort compared to the gains. We were also able to prove a theoretical improvement in the communication complexity from logarithmic to constant. In total, our algorithms reduce the wall-clock time of the largest simulated by approximately 78.8 %.

Our work is still three orders of magnitude away from a full-brain simulation. Scaling purely the number of MPI ranks seems infeasible (approximately 1.2 million MPI ranks with 65,536 neurons per rank), which leaves the simultaneous increase of neurons per MPI rank. The analysis of the time consumed by the largest simulation paints a clear picture for the road ahead: Due to our improvements, we reduced the communication overhead drastically. What remains is a growing portion of "per-neuron" work, which can be offloaded to a GPU—which will become advisable anyway, as today's HPC systems increasingly rely on GPUs to achieve performance. It remains to be seen how easily the repeated application of the Barnes–Hut algorithm we require can be mapped to the execution model of a GPU. If this succeeds, 131,072 MPI ranks (2^{17}) with approximately 600,000 neurons per rank could be feasible in the future—requiring 1261 compute nodes with 104 cores.

References

Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in human neuroscience*, 3:857, 2009.

Alain Goriely. Eighty-six billion and counting: do we know the number of neurons in the human brain? *Brain*, 148(3):689–691, 11 2024. ISSN 0006-8950. doi: 10.1093/brain/awae390.

Markus Butz and Arjen van Ooyen. A Simple Rule for Dendritic Spine and Axonal Bouton Formation Can Account for Cortical Reorganization after Focal Retinal Lesions. *PLoS Computational Biology*, 9(10):39–43, 2013. ISSN 1553734X. doi: 10.1371/journal.pcbi.1003259.

Sophie H Bennett, Alastair J Kirby, and Gerald T Finnerty. Rewiring the connectome: evidence and effects. *Neuroscience & Biobehavioral Reviews*, 88: 51–62, 2018.

Chiara La Rosa, Roberta Parolisi, and Luca Bonfanti. Brain structural plasticity: from adult neurogenesis to immature neurons. *Frontiers in Neuroscience*, 14:75, 2020.

Yana Fandakova and Catherine A. Hartley. Mechanisms of learning and plasticity in childhood and adolescence. *Developmental Cognitive Neuroscience*, 42:100764, 2020. ISSN 1878-9293. doi: https://doi.org/10.1016/j.dcn.2020.100764.

Ami Citri and Robert C Malenka. Synaptic plasticity: multiple forms, functions, and mechanisms. *Neuropsychopharmacology*, 33(1):18–41, 2008.

Jeffrey C Magee and Christine Grienberger. Synaptic plasticity forms and functions. *Annual review of neuroscience*, 43(1):95–117, 2020.

Markus Butz, Florentin Wörgötter, and Arjen van Ooyen. Activity-dependent structural plasticity. *Brain Research Reviews*, 60(2):287–305, 2009. ISSN 0165-0173. doi: https://doi.org/10.1016/j.brainresrev.2008.12.023.

Gaia Tavosanis. Dendritic structural plasticity. *Developmental neurobiology*, 72 (1):73–86, 2012.

Rei Yamada and Hiroshi Kuba. Structural and functional plasticity at the axon initial segment. *Frontiers in cellular neuroscience*, 10:250, 2016.

Júlia V. Gallinaro, Nebojša Gašparović, and Stefan Rotter. Homeostatic control of synaptic rewiring in recurrent networks induces the formation of stable memory engrams. *PLOS Computational Biology*, 18(2):1–40, 02 2022. doi: 10.1371/journal.pcbi.1009836.

Marvin Kaster, Fabian Czappa, Markus Butz-Ostendorf, and Felix Wolf. Building a realistic, scalable memory model with independent engrams using a homeostatic mechanism. *Frontiers in Neuroinformatics*, 18, 2024. ISSN 1662-5196. doi: 10.3389/fninf.2024.1323203.

Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

Sebastian Rinke, Markus Butz-Ostendorf, Marc André Hermanns, Mikaël Naveau, and Felix Wolf. A scalable algorithm for simulating the structural plasticity of the brain. *Journal of Parallel and Distributed Computing*, 120: 251–266, 2018. ISSN 07437315. doi: 10.1016/j.jpdc.2017.11.019.

Eric Darve. The fast multipole method: numerical implementation. *Journal of Computational Physics*, 160(1):195–240, 2000.

Hannah Nöttgen, Fabian Czappa, and Felix Wolf. Accelerating brain simulations with the fast multipole method. In *Proc. of the 28th Euro-Par Conference 2022: Parallel Processing, Glasgow, UK*, volume 13440 of *Lecture Notes in Computer Science*, pages 387–402. Springer, August 2022. ISBN 978-3-031-12597-3. doi: 10.1007/978-3-031-12597-3_24.

Fabian Czappa, Alexander Geiß, and Felix Wolf. Simulating structural plasticity of the brain more scalable than expected. *Journal of Parallel and Distributed Computing*, 171:24–27, January 2023. ISSN 0743-7315. doi: 10.1016/j.jpdc.2022.09.001.

Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*, November 2023. URL `https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf`.

Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

N. Abi Akar, B. Cumming, V. Karakasis, A. Küsters, W. Klijn, A. Peyser, and S. Yates. 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)*, pages 274–282, feb 2019. doi: 10.1109/EMPDP.2019.8671560.

Nicholas T Carnevale and Michael L Hines. *The NEURON book*. Cambridge University Press, 2006.

Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *elife*, 8:e47314, 2019.

Lars Niedermeier, Kexin Chen, Jinwei Xing, Anup Das, Jeffrey Kopsick, Eric Scott, Nate Sutton, Killian Weber, Nikil Dutt, and Jeffrey L Krichmar. Carlsim 6: an open source library for large-scale, biologically detailed spiking neural network simulation. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2022.

Paula Sanz Leon, Stuart A. Knock, M. M. Woodman, Lia Domide, Jochen Mersmann, Anthony R. McIntosh, and Viktor Jirsa. The virtual brain: a simulator of primate brain network dynamics. *Frontiers in Neuroinformatics*, Volume 7 - 2013, 2013. ISSN 1662-5196. doi: 10.3389/fninf.2013.00010.

Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.

Rajagopal Ananthanarayanan, Steven K. Esser, Horst D. Simon, and Dharmendra S. Modha. The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, 2009. doi: 10.1145/1654059.1654124.

Susanne Kunkel, Maximilian Schmidt, Jochen M. Eppler, Hans E. Plesser, Gen Masumoto, Jun Igarashi, Shin Ishii, Tomoki Fukai, Abigail Morrison, Markus Diesmann, and Moritz Helias. Spiking network simulation code for petascale computers. *Frontiers in Neuroinformatics*, Volume 8 - 2014, 2014. ISSN 1662-5196. doi: 10.3389/fninf.2014.00078.

Hiroshi Yamaura, Jun Igarashi, and Tadashi Yamazaki. Simulation of a human-scale cerebellar network model on the k computer. *Frontiers in Neuroinformatics*, Volume 14 - 2020, 2020. ISSN 1662-5196. doi: 10.3389/fninf.2020.00016.

Susanne Pfalzner and Paul Gibbon. *Many-Body Tree Methods in Physics*. Cambridge University Press, 1996.

A. Costa, U. Becciani, P. Miocchi, V. Antonuccio, R. Capuzzo Dolcetta, P. Di Matteo, and V. Rosato. Astrocomp: web technologies for high performance computing on a network of supercomputers. *Computer Physics Communications*, 166(1):17–25, 2005. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2004.09.010.

Joshua E Barnes. A modified tree code: don't laugh; it runs. *Journal of Computational Physics*, 87(1):161–170, 1990.

Martin Burtscher and Keshav Pingali. An efficient cuda implementation of the tree-based barnes hut n-body algorithm. In *GPU computing Gems Emerald edition*, pages 75–92. Elsevier, 2011.

V Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60(2):187–207, 1985. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(85)90002-6.

Gianluca Ceruti, Jonas Kusch, and Christian Lubich. A parallel rank-adaptive integrator for dynamical low-rank approximation. *SIAM Journal on Scientific Computing*, 46(3):B205–B228, 2024. doi: 10.1137/23M1565103.

Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1995.1039.

Salim Rukhsar and Anil Kumar Tiwari. Barnes–hut approximation based accelerating t-sne for seizure detection. *Biomedical Signal Processing and Control*, 84:104833, 2023. ISSN 1746-8094. doi: https://doi.org/10.1016/j.bspc.2023.104833.

Darian Hadjiabadi, Matthew Lovett-Barron, Ivan Georgiev Raikov, Fraser T Sparks, Zhenrui Liao, Scott C Baraban, Jure Leskovec, Attila Losonczy, Karl Deisseroth, and Ivan Soltesz. Maximally selective single-cell target for circuit control in epilepsy models. *Neuron*, 109(16):2556–2572, 2021.

E.M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003. doi: 10.1109/TNN.2003.820440.