Accelerating Brain Simulations with the Fast Multipole Method

Hannah Nöttgen[1,2][0000−0001−6481−5717], Fabian Czappa[1,3][0000−0001−8422−5706], and Felix Wolf[1,4][0000−0001−6595−3599]

[1] Laboratory for Parallel Programming, Technical University of Darmstadt, Darmstadt, Germany
[2] hannah.noettgen@stud.tu-darmstadt.de
[3] fabian.czappa@tu-darmstadt.de
[4] felix.wolf@tu-darmstadt.de

Abstract. The brain is probably the most complex organ in the human body. To understand processes such as learning or healing after brain lesions, we need suitable tools for brain simulations. The Model of Structural Plasticity offers a solution to that problem. It provides a way to model the brain bottom-up by specifying the behavior of the neurons and using structural plasticity to form the synapses. However, its original formulation involves a pairwise evaluation of attraction kernels, which drastically limits scalability. While this complexity has recently been decreased to $O(n \cdot \log^2 n)$ after reformulating the task as a variant of an n-body problem and solving it using an adapted version of the Barnes–Hut approximation, we propose an even faster approximation based on the fast multipole method (FMM). The fast multipole method was initially introduced to solve pairwise interactions in linear time. Our adaptation achieves this time complexity, and it is also faster in practice than the previous approximation.

Keywords: Fast Multipole Method · Brain Simulation · Structural Plasticity · Scalability

1 Introduction

The human brain undergoes constant change not only in children but throughout the whole life [8]. These changes, especially in the form of synapse creation and deletion, are believed to be responsible for a major portion of the brain dynamics. There is overwhelming evidence that structural plasticity, i.e., the change of connectivity of neurons, is responsible for learning, memory creation, and healing after lesions [12, 23, 20, 24, 25]. However, current in vivo imaging techniques cannot create connectivity maps for human brains at a scale comparable to the original [5, 12, 13]. This leaves a large portion of current research in need of simulations to fill the gap. Many state-of-the-art simulators can mimic very complex behaviors of a single neuron, however, they lack the possibility to let neurons freely connect to others. This task inherently involves solving pairwise interactions. Seeing that

the human brain contains 86 billion neurons [6], this drastically limits scalability. Many simulators bypass this issue by only allowing already existing connections to be strengthened or weakened (synaptic plasticity), bringing the complexity down to linear in the number of neurons and synapses per neuron.

It is not fully understood how neurons form and delete synapses over time. For a long time, Hebbian Plasticity [19] was the dominant opinion. In recent times, however, homeostatic mechanisms—in which neurons pursue a stable state and thus the whole brain reaches an equilibrium—have been suggested and shown to be accurate [7, 10]. One of these mechanisms, the Model of Structural Plasticity [7], predicts the recovery of lesions in mice very well. In a recent publication, Rinke et al. [27] have reduced the quadratic complexity of solving the pairwise interactions to $O(n \cdot \log^2 n)$. They achieved this by utilizing the Barnes–Hut algorithm [3], which has been developed to approximately solve pairwise interactions, which is popular in the context of physics.

We propose another approximation for the pairwise interactions based on the fast multipole method (FMM) [28]. While Barnes–Hut calculates point-area interactions, FMM calculates area-area interactions, reducing the complexity from quadratic to linear. Current in vivo imaging techniques such as [9] cannot precisely locate where synapses begin and end; they can only trace them to a certain area. This, together with the fact that we do not know exactly why a particular neuron formed a (long-reaching) synapse and not its direct neighbor, gives us confidence that this approximation is reasonable.

In this publication, we build on the Barnes–Hut approximation and utilize their distributed algorithm to implement our approximation in terms of the fast multipole method. To summarize, our main contributions are:

- We integrated the fast multipole method into an existing parallel neuron simulation and replaced the Barnes–Hut algorithm, which was responsible for finding synapses.
- We reduced the theoretical complexity from $O(n/p \cdot \log^2 n)$ to $O(n/p + p)$, when n is the number of input neurons and p the number of MPI ranks.
- We measured the influence on performance in practice for different numbers of computing nodes.

The remainder of this paper is structured as follows. We firstly review related work in Section 2 before we explain relevant background in Section 3. Afterward, we present our algorithm in Section 4, and analyze it in terms of theoretical and practical run time with multiple compute nodes in Section 5.

2   Related Work

There are many brain simulators freely available, for example, C2 [2], NEST [16], and The Virtual Brain [29]. They allow initial connectivity of neurons to be inserted, and during the simulation, they may strengthen and weaken those. However, they do not create new connections.
This way, their connectivity update step has complexity $O(n \cdot m)$, where n is the

number of neurons and m is the number of synapses per neuron. The latter term is most often bounded (e.g., at most 1000 synapses per neuron), and thus, it is linear in the number of neurons.

Structural plasticity—the way in which neurons grow new and delete old synapses—has gained track in recent years; see [31] for an overview of the current state of the art. A simple model for structural plasticity, proposed by van Ooyen et al. [30], achieves this by defining the outreach of a neuron to be a circle around its center. The individual neurons form synapses proportional to the overlapping area whenever two such circles overlap. This model requires linear time for the connectivity update, but it lacks the possibility of connecting neurons while omitting a third neuron between them. The Model of Structural Plasticity (MSP) [7], on which this publication is based, overcomes this limitation by calculating the connection probability dependent on the distance of neurons.

In the MSP, the likelihood of a synapse forming between two neurons with positions p_1, p_2, is proportional to $\texttt{exp}(-||p_1 - p_2||_2^2/\sigma)$, with a scaling constant $\sigma > 0$. This way, the greater the distance between two neurons is, the smaller the likelihood of a connection between them is. Rinke et al. [27] used this insight to approximate the influence of a whole area of neurons far away with the Barnes–Hut algorithm [3]. They achieved this by inserting the neurons into an octree and calculating the attraction to inner nodes whenever possible (thus skipping the need to calculate the attraction to all neurons in the induced subtree). This way, they reduced the complexity of calculating $O(n^2)$ interactions (n again the number of neurons) to $O(n \cdot \log^2 n)$.

The fast multipole method (FMM) [17, 28] is another way of approximating pairwise interactions. Instead of only combining the affecting elements (neurons, particles, etc.) at the calculation's source side, they also group the affected elements at the target side. This way, they can approximate the pairwise interactions in linear time using Hermite and Taylor expansions [14, 15]. This is used quite successfully in physics, including astrophysics [11] and particle simulation [1]. There exists many accelerated FMM implementations both with GPUs and MPI, for example [18, 26, 32]. However, they focus on a fixed-level attraction, i.e., in contrast to us, they don't need to resolve the attractions down to a object–object level.

3 Background

In this section, we repeat the arguments and definitions from previous publications to be partially self-contained. This includes the initial publication for the Model of Structural Plasticity [7], the one that introduced the fast multipole method we use [17], and the publication that introduced the Barnes–Hut approximation to the MSP [27].

3.1 The Model of Structural Plasticity

The Model of Structural Plasticity [7] describes how neurons change their plasticity over time, i.e., how they form new and delete old synapses. It consists of

three different phases:
The update of electrical activity, the update of synaptic elements, and the update of synapses. An overview of the used model parameters can be seen in Table 1.

Update of Electrical Activity In the step to update the electrical activity, all neurons calculate their current activity. This can be done by neuron models such as the one proposed by Izhikevich [21], the FitzHugh–Nagumo model [22], or as in our case, a Poisson spiking neuron model (the same as in [27]).

In our model, the activity, on the one hand, strives exponentially to a resting potential (resting potential: 0.05, constant of decay: 5); on the other hand, it is constantly increased by a small background activity (0.003) and the input of all connected neurons (those neurons that form a synapse from their axon to the dendrite of the neuron in question) that spiked in the last update step by a fixed amount (5e-4). Then a uniformly distributed value from $[0, 1]$ is drawn, and if this value is smaller than the current activity, the neuron spikes. If a neuron spikes, it does not spike again for a fixed number of steps (refractory period: 4).

Update of Synaptic Elements In the update step, each neuron updates its intercellular calcium level. The calcium level decays exponentially (constant of decay: 1e-5), while if a neuron spiked in that simulation step, it is increased by a fixed value (1e-3).

After the neuron has updated its calcium, it uses this to determine the amount of change to its synaptic elements. We use the same Gaussian growth curve as originally proposed in [7], setting the right intersection (the target value) to 0.7, the left intersection (the point at which the elements start to grow) to 0.4 for axons, and 0.1 for dendrites, and the scaling parameter to 1e-4 (maximum attained value). The neuron updates the number of axons and dendrites by the calculated amount.

Update of Synapses Every time a neuron updates its synapses (once every 100 updates of activity and synaptic elements), it checks its number of synaptic elements. If it now has fewer elements than synapses (the elements are continuous, the synapses discrete), it chooses synapses randomly, notifies the connected neurons, and deletes them. It does so for both the dendrites with the incoming synapses and the axons with the outgoing synapses.

After the deletion phase, if a neuron has at least one vacant axon, it searches for another neuron with one vacant dendrite to connect to. For each vacant axon (i), it calculates the probability of connecting to a vacant dendrite (j) by

$$K(i,j) = \exp\Big(\frac{-||\texttt{pos}_i - \texttt{pos}_j||_2^2}{\sigma}\Big) \tag{1}$$

and chooses one such vacant dendrite ($\sigma = 750$ as in [7]). These requests are gathered (this takes quadratic time) and sent to the neurons with vacant dendrites. Those resolve potential conflicts, and a new synapse is formed whenever possible.

Name	Symbol	Value
Resting potential	x_0	0.05
Membrane potential constant of decay	τ_x	5
Background activity	I	0.003
Increase in calcium per spike	β	5e-4
Calcium constant of decay	τ_{Ca}	1e-5
Gaussian growth curve right intersection	ϵ	0.7
Gaussian growth curve left intersection (axons)	η_A	0.4
Gaussian growth curve left intersection (dendrites)	η_D	0.1
Growth scaling parameter	μ	1e-4
Probability kernel standard deviation	σ	750

Table 1: Overview of the model parameters used in the executions and tests. A more detailed description of the model and the parameters can be found in [7].

3.2 A Distributed Octree

In [27], Rinke et al. introduced a distributed octree to overcome the memory limitations inherent to large simulations. The problem is that only a limited number of octree nodes can be held in memory. In order to be able to simulate more neurons and thus achieve the desired order of magnitude, many MPI ranks are required. They recursively divide the simulation domain into eight cells until a cell contains at most one neuron. Inner nodes of the octree store the sum of vacant elements of all their children and the combined position (the centroid), which is just the weighted average position of the children. The octree is updated in a step-wise fashion: All ranks update their subtrees, then exchange the branch nodes, and calculate the shared upper portion afterward. They insert all neurons into a spatial octree, where every MPI rank is responsible for 1, 2, or 4 subtrees. All ranks share the same upper portion of the octree (heights 0 to $\log(8, p)$ where p is the number of ranks), and if a rank i requires information of a neuron on rank j, it downloads them lazily.

3.3 Mathematical Formulation of the Fast Multipole Method

Assuming there is a set of points in space, we consider a split into M sources $s_1, \ldots, s_M$ (in our case the neurons that have a vacant dendrite) and N targets (the neurons that have a vacant axon) $t_1, \ldots, t_N$. If a neuron has vacant axons as well as vacant dendrites, it is included in both sets M and N. The general form of an n-body problem is [4]:

$$u(t) = \sum_{i=1}^{M} \omega_i \cdot K(t, s_i) \tag{2}$$

$\omega_i \in \mathbb{R}$: The weight of the point s_i.

$K : (\mathbb{R}^3 \times \mathbb{R}^3) \to \mathbb{R}$: A kernel that calculates the interaction between t and s_i.

This formula gives the total attraction $u(t)$ for a vacant axon. If a neuron has more than one vacant axon, we multiply $u(t)$ with that number. In order to calculate $u(\cdot)$ for every target t_j, this function must be calculated N times, which results in a total complexity of $O(N \cdot M)$. If every source is also a target, i.e., $N = M$, this scales quadratically.

The general form of Equation 2 fits the attraction formula of the MSP (cf. Equation 1), with ω_i being the number of vacant dendrites of a neuron and $u(t)$ being the force of attraction to an axon of a neuron at position t.

Notation A multi-index $\alpha = (n_1, n_2, n_3)$ is a tuple of three natural numbers (including zero). For any multi-index and any vector $t = (x, y, z) \in \mathbb{R}^3$, we define the following operations:

$$|\alpha| = n_1 + n_2 + n_3 \tag{3}$$

$$\alpha! = n_1! \cdot n_2! \cdot n_3! \tag{4}$$

$$t^\alpha = x^{n_1} \cdot y^{n_2} \cdot z^{n_3} \tag{5}$$

For our adapted fast multipole method we often use multi-indices in combination with sums. For example, $\sum_{\alpha \geq p}$ or $\sum_{0 \leq \alpha \leq p}$ stands for three nested sums with $n_1, n_2, n_3 \geq p$ or $0 \leq n_1, n_2, n_3 \leq p$, respectively.

Approximations of Attraction Kernel In general the MSP sums over Gaussian functions. We can approximate the attraction of multiple neurons in a box and group the sources and targets together. For each such box (S for a box of sources, T for a box of targets), we need to calculate the centroid with respect to its sources s_C and its targets t_C. Using the function $h(\alpha, x)$ (the same as in [17] Equation 8) and $\delta = \sigma^2$, Equation 2 with the Gaussian kernel of Equation 1 has the following Taylor series (using a multi-index β):

$$
\begin{aligned}
u(t) &= \sum_{0 \leq \beta} B_\beta \cdot \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \\
B_\beta &= \frac{(-1)^{|\beta|}}{\beta!} \cdot \sum_{j=1}^{M} \omega_j \cdot h\left(\beta, \frac{s_j - t_C}{\sqrt{\delta}} \right)
\end{aligned}
\tag{6}
$$

We can truncate the outer series from Equation 6, i.e., sum only up to some fixed β. The approximation error depends on the box side length, which is determined by the box's level in the octree. Furthermore, the number of calculated terms and the number of sources that are approximated have an influence on the approximation error. Overall, the calculation of one coefficient B_β from Equation 6 has complexity $O(M)$, and crucially they are shared for all targets in the box. In addition, Equation 6 must be calculated for N target points with k coefficients. This results in a complexity of $O(k \cdot M + k \cdot N)$ for a interaction between one target box and one source box.

Alternatively, we can also approximate Equation 1 with Hermite coefficients A_α, where the same argument as before applies (with multi-index α). The complexity to calculate this expansion is also $O(k \cdot M + k \cdot N)$ for k coefficients:

$$u(t) = \sum_{0 \leq \alpha} A_\alpha \cdot h\left(\alpha, \frac{t - s_C}{\sqrt{\delta}}\right)$$

$$A_\alpha = \frac{1}{\alpha!} \cdot \sum_{j=1}^{M} \omega_j \cdot \left(\frac{s_j - s_C}{\sqrt{\delta}}\right)^\alpha \tag{7}$$

4   Algorithm Description

To determine which neurons form synapses with each other, we must calculate the forces of attraction between the target and source neurons (note here that a "source" and "target" are used differently in the literature: The "source" neuron is the one with the axon, however, it is the "target" of the attraction). Therefore, we create an n-body problem on top of the kernel in Equation 1 in order to apply the series expansions already presented:

$$u(t) = \sum_{j=1}^{M} \omega_j \cdot \exp\left(\frac{-||t - s_j||_2^2}{\sigma^2}\right), \tag{8}$$

where ω_j is the number of vacant dendritic elements of the j-th neuron. Furthermore, we use the same distributed octree as in [27]. In our version—compared to the Barnes–Hut inspired one—we also need to calculate the centroid of the inner nodes with respect to the axons. For this, we increased the size of the octree nodes from 200 Bytes to 264 Bytes (2x 32 Byte for the axon positions, which consist of 24 Bytes vector and a flag).

Algorithm 1 shows the implementation for finding suitable neurons. For the initialization of the stack (Line 2), we first collect all roots of the subtrees and then find another subtree-root as the target for each of them, as described in the paragraph below. We push the source–target pairs onto the stack and then process the elements of the stack until it is empty. Whenever we want to form a synapse, we save the source and target ids and send them to the MPI rank of the target. Each rank collects these requests, chooses locally which to accept (to avoid too many synapses, e.g., five axons want to connect to two dendrites), and sends the answers back.

Initialization of the Stack Every MPI rank does the following: For each of its subtree-roots, choose a target with `choose_target` (cf. Algorithm 2) with the global root as initial target. Fix the subtree-root and only unpack the targets until the target is at the same level as the subtree-root. Put these pairs on the stack.

```
1  find_synapses():
2      init(stack)
3
4      while (! stack.empty()):
5          source_node, target_node = stack.pop()
6
7          if (source_node.is_leaf && target_node.is_leaf):
8              form_synapse(source_node, target_node)
9
10         else if (source_node.is_leaf):
11             new_target = choose_target(source_node, target_node.children)
12             stack.push(source_node, new_target)
13
14         else if (target_node.is_leaf):
15             new_source = choose_source(source_node.children, target_node)
16             stack.push(new_source, target_node)
17
18         else: foreach (new_source in source_node.children)
19                 new_target = choose_target
20                     (new_source, target_node.children)
21                 stack.push(new_source, new_target)
```

Algorithm 1: Pseudo code of the method `find_synapses`. `choose_target` and `choose_source` are shown in Algorithm 2.

Choice of Target Node This method calculates the attractiveness of the target neurons to the source neuron. It does so by first determining if it needs to evaluate the formula directly or if it can use an approximation (Taylor or Hermite). It then calculates the attractiveness of the children of the target and chooses one randomly, proportional to their attractiveness. In addition, the method needs two constants `c1` and `c2`, which determine when a Taylor or a Hermite expansion is used as it is easier to evaluate the attractions directly if the number of dendrites and axons is small.

4.1   Complexity

For the complexity, it is enough to determine how often `choose_target` is called. We assume a balanced octree and start with the serial version. Starting at level 0, the root must determine a target for each of its children, so it calls `choose_target` 8 times and thus spawns 8 new pairs to consider. For each of the newly created pairs, the same applies, they spawn (up to) 8 new tasks, and in general, processing level k of the octree spawns 8^k tasks. As the tree is balanced its height is $\log(8, n)$ for n neurons, so it spawns $8, 64, \ldots, n/8, n$ tasks, i.e., linear in the number of neurons.

`choose_target` itself either performs a direct pair-wise calculation (with quadratic complexity) or one of the FMM approximations (with linear complexity). In our instance, however, `choose_target` has a constant complexity because the number of sources and targets is at most 8.

For the parallel version, we have to initialize the stack first. This requires choosing a target node for each subtree-root on the same level, which is $\log(p)$ for p the number of MPI ranks. Once we have found the pairs, we can apply the serial version to n/p neurons, so the overall complexity is $O(n/p + \log p)$. Gathering the branch nodes beforehand has complexity $O(p)$, so the overall complexity of the connectivity update is $O(n/p + p)$.

```
1   choose_target(source_node, target_node):
2       probabilities = [ ]
3
4       for (i = 0; i < target_node.number_children; i++):
5           child = target_node.children[ i ]
6
7           if (source.is_leaf || child.is_leaf):
8               probabilities[ i ] = direct_calculation()
9
10          else if (child.get_number_dendrites() > c1
11             && source_node.get_number_axons() > c2):
12              probabilities[ i ] = calculate_hermite_expanison()
13
14          else if (child.get_number_dendrites() > c1):
15              probabilities[ i ] = calculate_taylor_expanison()
16
17          else:
18              probabilities[ i ] = direct_calculation()
19
20      total_probability = sum(probabilities)
21      rand = uniform_random(0, total_probability)
22      index = upper_bound(probabilities, rand)
23
24      return target_node.children[index]
```

Algorithm 2: `choose_target` calculates the probability for `source_node` to connect to each child of `target_node`. It chooses the method based on the number of vacant axons and dendrites. It then picks one target neuron randomly with chances proportional to the calculated probabilities. `choose_source` works analogously with swapped roles of `source_node` and `target_node`. In our case, `c1 = 70` and `c2 = 70`.

5   Evaluation

Our proposed algorithm trades time against freedom of choice; the complexity of finding the target neurons is lower than for the adapted Barnes–Hut algorithm, however, we lose some freedom of choice for the synapses. Previously, with the Barnes–Hut algorithm, each axon searched its own dendrite, so if a neuron had two vacant axons, they could connect to dendrites with a large distance between them as they could be in different nodes. In our algorithm, both axons are always in the same box, so their choice will be the same throughout the whole process. This also affects axons on neurons that are close to each other—if they are in the same box on level l, their choice of boxes coincides on every level $i = 0, \ldots, l-1$. This, in turn, means that every neural network that was calculated with the fast multipole method can also be calculated with the Barnes–Hut algorithm, but not vice versa.

All calculations for this research were conducted on the Lichtenberg 2 high-performance computer of the TU Darmstadt. One compute node has 2 Intel Xeon Platinum 9242 processors (with disabled hyper-threading), 384 GB main memory, and the interconnection is a 100 GBit/s InfiniBand. We always tested the algorithm with 500 000 simulation steps (5000 connectivity updates) and a network with only excitatory neurons.

From a neuroscientific point of view, we investigated the following observable metrics for both the Barnes–Hut and the fast multipole method inspired algorithms:

1. The average calcium concentration of the neurons (together with the standard deviation) to see how well both algorithms allow the neurons to reach a local equilibrium.
2. The number of formed synapses to see how well the overall simulation reaches a global equilibrium.

Figure 1 shows the average calcium of the neurons (together with its standard deviation), and Figure 2 shows the total number of created synapses for one run of $p = 64$ MPI ranks and $n = 320\,000$ neurons. With our proposed algorithm, the average calcium is nearly indistinguishable from the Barnes–Hut algorithm, however, its standard deviation is slightly higher. Our algorithm trails the previous version slightly when it comes to the total number of formed synapses. This is due to more collisions, resulting in more rejections, so we need more simulation steps to connect all vacant axons. Furthermore, the total number of synapses is less for our algorithm. The reason is that a neuron generally grows more dendrites than axons. If the synapses now cluster more (due to the restricted freedom of choice), some neurons receive more synapses than they want—so they delete some again.
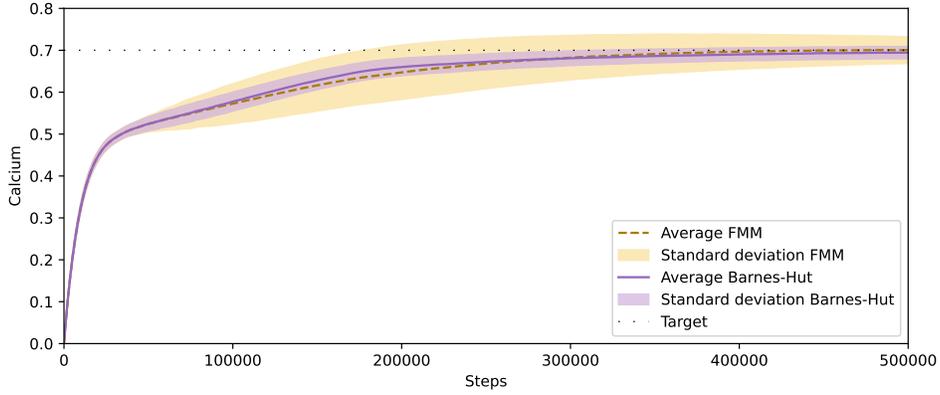
Fig. 1: The average calcium (solid lines) and its standard deviation (light area) for both algorithms (Barnes–Hut in solid purple, fast multipole method in dashed orange). The target calcium is 0.7 (thinly dashed black line). $p = 64$ MPI ranks, $n = 320\,000$ neurons, and $500\,000$ simulation steps (5000 connectivity updates).
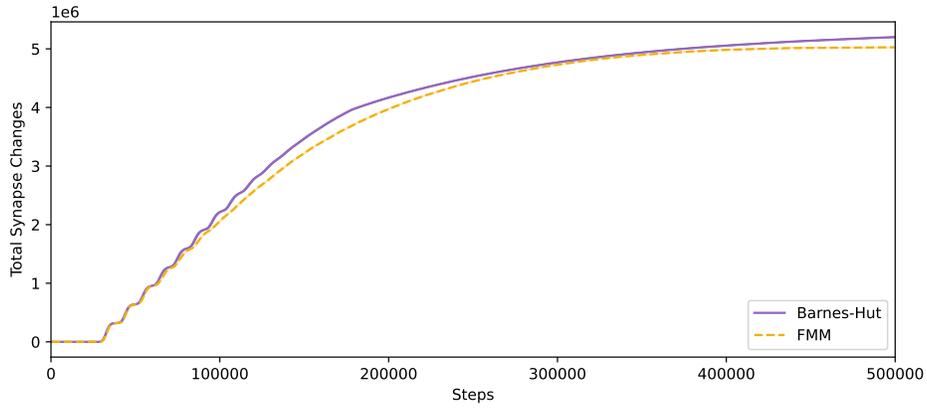

Fig. 2: The total number of synapses for both algorithms. $p = 64$ MPI ranks, $n = 320\,000$ neurons, and $500\,000$ simulation steps (5000 connectivity updates).
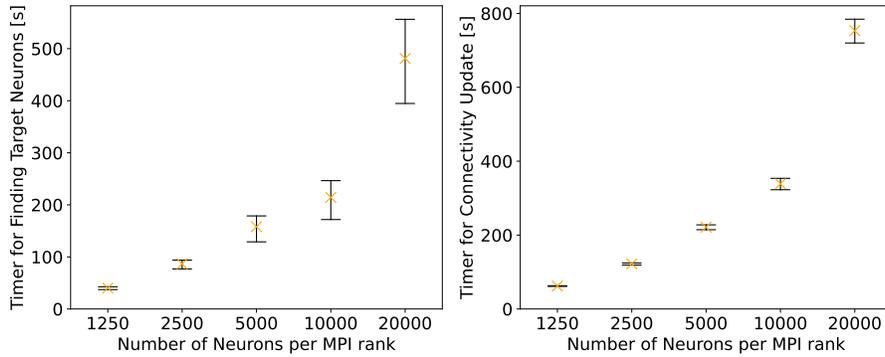
Fig. 3: The timings for the strong scaling experiments with $p = 64$ MPI ranks and $500\,000$ simulation steps (5000 connectivity updates). We give the minimum, average, and maximum time across the different ranks. All timings are in seconds.

We evaluated the strong-scaling behavior of our algorithm with $p = 64$ MPI ranks and $n = 1\,250, 2\,500, 5\,000, 10\,000, 20\,000$ excitatory neurons per MPI rank. Figure 3 shows the minimum, average, and maximum time for a simulation across all MPI ranks. We have conducted these simulations five times and found that the generated timings are very stable concerning the repetitions. When calculating the coefficient of variation (standard deviation divided by average) for these repetitions, it is consistently below 1%. Doubling the number of neurons per rank scales the time of the connectivity update by approximately 1.96, 1.81, 1.53, and 2.22, and the time it takes to find the targets by 2.15, 1.82, 1.35, and 2.25. Overall, these timings suggest to us a good strong-scaling behavior.

Furthermore, we tested the weak-scaling behavior of our algorithm with $n = 5000$ neurons per MPI rank (as in the previous publication which introduced the Barnes–Hut approximation [27]) and with $p = 1, 2, 4, 8, 16, 32, 64$ MPI ranks. For the timings, we investigated the overall time for the connectivity update, the time it takes to find the target neurons, and for the fast multipole method also the time it takes to compute the expansions. Figure 4 shows the minimum, average, and maximum time for a simulation across all MPI ranks. We have repeated each measurement five times with no significant difference. This means that the coefficient of variation remained below 1% in this experiment as well. Between one tenth and one third of the time the fast multipole method spends finding the synapses is spent in the Taylor expansion; the Hermite expansion is rarely used. The difference between the MPI ranks is low in our algorithm compared to the Barnes–Hut algorithm. Besides network communication noise, this difference is caused by neurons choosing partners close to or far away from others. Per connectivity update, we cache the already fetched octree nodes from other MPI ranks. This way, our algorithm profits from the locality of target choices compared to the Barnes–Hut algorithm. Overall, the new connectivity update is significantly faster, and the scaling behavior fits the broad expectation of $O(n/p + p)$.

Lastly, we evaluated the influence of the parameters β from Equation 6 and α from Equation 7, i.e., the points at which we cut of the evaluation of the infinite series. For this, we have conducted 12\,188 representative calculations for each expansion, as well as the direct evaluation. Figure 5 displays the results, showing that our cut-off point with $\alpha = \beta = (3, 3, 3)$ is well chosen and more terms do not enhance the accuracy significantly.
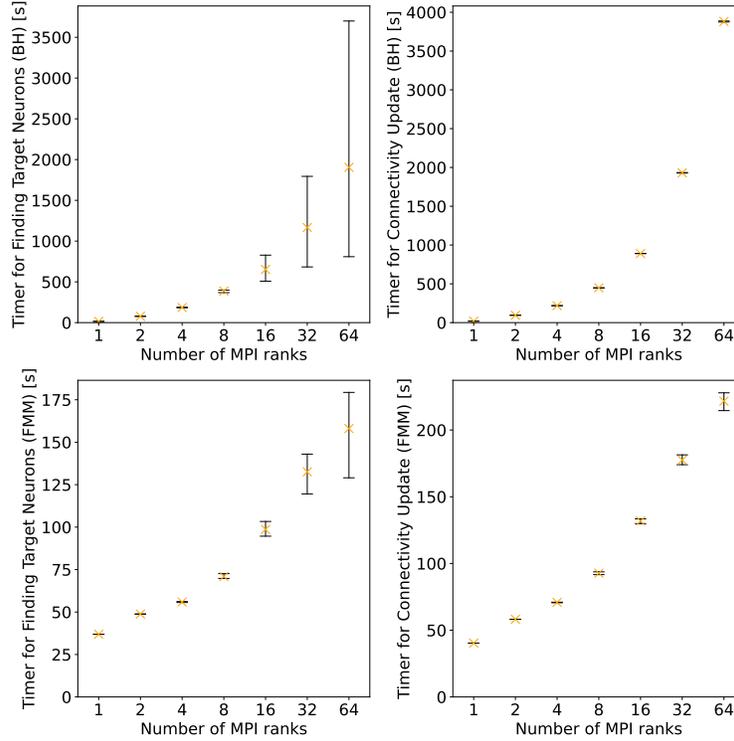
Fig. 4: The timings of different methods of the simulation for $p = 1, 2, 4, 8, 16, 32, 64$ MPI ranks and $500\,000$ simulation steps (5000 connectivity updates).. For each method we give the minimum, average, and maximum time across the different MPI ranks. All timings are in seconds.
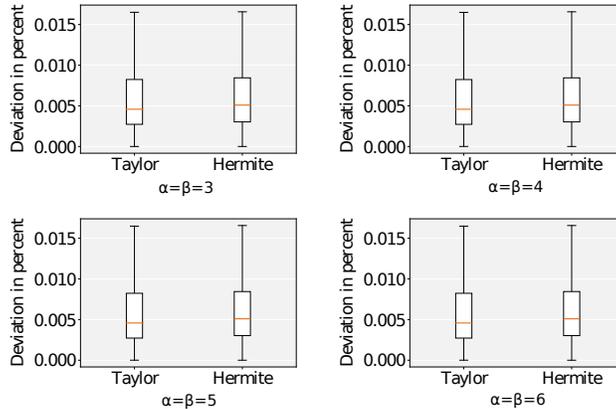

Fig. 5: The deviation in percent between the directly evaluated attraction and the corresponding Hermite and Taylor expansions, gathered from $12\,188$ representative boxes. The red line is the median, the box indicates the 0.25 and 0.75 quartile, the interval indicates the minimum and maximum after removing all outliers. A value is an outlier if it is larger then the 0.75 quartile + 1.5 times the inter-quartile range. The number of outliers for the Taylor expansions were 1830, 1834, 1833, and 1833, and there were consistently 1753 outliers for the Hermite expansions. The largest outliers were below 0.125 %.

6 Conclusion

This work aimed to replace the Barnes–Hut algorithm in an existing neuron simulation with multiple computing nodes with the fast multipole method of lower complexity and to measure the influence on performance. We achieved a theoretical complexity of $O(n/p + p)$, when n is the number of input neurons and p the number of MPI ranks, which is lower than the previous complexity of $O(n/p \cdot \log^2 n)$. In addition, the algorithm presented here is faster in practice on multiple computing nodes, exhibits a good strong-scaling behavior, and additionally, the rank-to-rank variation shrank significantly. Also, the internal calcium concentration and the formation of synapses behave very closely to the original simulation. However, there are aspects in which the algorithm presented here is inferior to the Barnes–Hut algorithm. The storage space consumption has increased by 32 %, and the choices of neighboring neurons are now more similar than before. In addition, our algorithm needs more simulation steps to connect all vacant elements through synapses due to more collisions.

In the future, we seek to combine the variable precision of the Barnes–Hut algorithm with our proposed one, which might let neurons form connections more independently than their neighbors. Furthermore, we plan to analyze the resulting networks with respect to the graph-topological metrics so we can assess the functionality of the networks.

Acknowledgments

References

1. Ambrosiano, J., Greengard, L., Rokhlin, V.: The fast multipole method for gridless particle simulation. Computer Physics Communications **48**(1), 117–125 (1988). https://doi.org/10.1016/0010-4655(88)90029-X
2. Ananthanarayanan, R., Esser, S.K., Simon, H.D., Modha, D.S.: The cat is out of the bag: Cortical simulations with 109 neurons, 1013 synapses. SC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1654059.1654124

3. Barnes, J., Hut, P.: A hierarchical o (n log n) force-calculation algorithm. nature **324**(6096), 446–449 (1986). https://doi.org/10.1038/324446a0

4. Beatson, R., Greengard, L.: A short course on fast multipole methods. Tech. rep., Department of Mathematics and Statistics, University of Canterbury and Courant Institute of Mathematical Sciences, New York University (1997)

5. Beaujoin, J., Palomero-Gallagher, N., Boumezbeur, F., Axer, M., Bernard, J., Poupon, F., Schmitz, D., Mangin, J.F., Poupon, C.: Post-mortem inference of the human hippocampal connectivity and microstructure using ultra-high field diffusion mri at 11.7 t. Brain Structure and Function **223**(5), 2157–2179 (2018). https://doi.org/10.1007/s00429-018-1617-1

6. Berlin, C.U.: Brain simulation section. https://www.brainpreservation.org/content-2/connectome/, visited 06.2021, (2019)

7. Butz, M., van Ooyen, A.: A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions. PLOS Computational Biology **9**(10), 1–21 (10 2013). https://doi.org/10.1371/journal.pcbi.1003259

8. Butz, M., Wörgötter, F., Ooyen, A.: Activity-dependent structural plasticity. Brain research reviews **60**, 287–305 (02 2009). https://doi.org/10.1016/j.brainresrev.2008.12.023

9. Chen, X., Wang, Y., Kopetzky, S.J., Butz-Ostendorf, M., Kaiser, M.: Connectivity within regions characterizes epilepsy duration and treatment outcome. Human Brain Mapping **42**(12), 3777–3791 (2021). https://doi.org/10.1002/hbm.25464

10. Dammasch, I.E., Wagner, G.P., Wolff, J.R.: Self-stabilization of neuronal networks. Biological Cybernetics **54**(4), 211–222 (8 1986). https://doi.org/10.1007/BF00318417

11. Dehnen, W.: A fast multipole method for stellar dynamics. Computational Astrophysics and Cosmology **1**(1), 1 (2014). https://doi.org/10.1186/s40668-014-0001-7

12. Diaz-Pier, S., Naveau, M., Butz-Ostendorf, M., Morrison, A.: Automatic generation of connectivity for large-scale neuronal network models through structural plasticity. Frontiers in Neuroanatomy **10**, 57 (2016). https://doi.org/10.3389/fnana.2016.00057

13. Dodt, H.U., Leischner, U., Schierloh, A., Jährling, N., Mauch, C.P., Deininger, K., Deussing, J.M., Eder, M., Zieglgänsberger, W., Becker, K.: Ultramicroscopy: three-dimensional visualization of neuronal networks in the whole mouse brain. Nature methods **4**(4), 331–336 (2007). https://doi.org/10.1038/nmeth1036

14. Eylert, B.: Praktische Mathematik für Informatiker, Telematiker und Ingenieure. Wildau (2014)

15. Friedrich, H., Pietschmann, F.: Numerische Methoden : ein Lehr- und Übungsbuch. De Gruyter Studium, Berlin, 2. auflage edn. (2020)

16. Gewaltig, M.O., Diesmann, M.: Nest (neural simulation tool). Scholarpedia **2**(4), 1430 (2007)

17. Greengard, L., Strain, J.: The fast gauss transform. SIAM Journal on Scientific and Statistical Computing **12**(1), 79–94 (1991). https://doi.org/10.1137/0912004

18. Gumerov, N.A., Duraiswami, R.: Fast multipole methods on graphics processors. Journal of Computational Physics **227**(18), 8290–8313 (2008). https://doi.org/10.1016/j.jcp.2008.05.023

19. Hebb, D.O.: The organization of behavior: A neuropsychological theory. Psychology Press (2005)

20. Holtmaat, A., Svoboda, K.: Experience-dependent structural synaptic plasticity in the mammalian brain. Nature Reviews Neuroscience **10**(9), 647–658 (9 2009). https://doi.org/10.1038/nrn2699

21. Izhikevich, E.M.: Simple model of spiking neurons. IEEE Transactions on neural networks **14**(6), 1569–1572 (2003). https://doi.org/10.1109/TNN.2003.820440
22. Izhikevich, E.M., FitzHugh, R.: Fitzhugh-nagumo model. Scholarpedia **1**(9), 1349 (2006)
23. Keck, T., Mrsic-Flogel, T.D., Vaz Afonso, M., Eysel, U.T., Bonhoeffer, T., Hübener, M.: Massive restructuring of neuronal circuits during functional reorganization of adult visual cortex. Nature Neuroscience **11**(10), 1162–1167 (10 2008). https://doi.org/10.1038/nn.2181
24. Kleim, J.A., Freeman, J.H., Bruneau, R., Nolan, B.C., Cooper, N.R., Zook, A., Walters, D.: Synapse formation is associated with memory storage in the cerebellum. Proc. of the National Academy of Sciences **99**(20), 13228–13231 (2002). https://doi.org/10.1073/pnas.202483399
25. Kleim, J.A., Hogg, T.M., VandenBerg, P.M., Cooper, N.R., Bruneau, R., Remple, M.: Cortical synaptogenesis and motor map reorganization occur during late, but not early, phase of motor skill learning. Journal of Neuroscience **24**(3), 628–633 (2004). https://doi.org/10.1523/JNEUROSCI.3440-03.2004
26. Lashuk, I., Chandramowlishwaran, A., Langston, H., Nguyen, T.A., Sampath, R., Shringarpure, A., Vuduc, R., Ying, L., Zorin, D., Biros, G.: A massively parallel adaptive fast-multipole method on heterogeneous architectures. In: Proc. of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 1–12 (2009). https://doi.org/10.1145/1654059.1654118
27. Rinke, S., Butz-Ostendorf, M., Hermanns, M.A., Naveau, M., Wolf, F.: A scalable algorithm for simulating the structural plasticity of the brain. Journal of Parallel and Distributed Computing **120**, 251–266 (2018). https://doi.org/10.1016/j.jpdc.2017.11.019
28. Rokhlin, V.: Rapid solution of integral equations of classical potential theory. Journal of Computational Physics **60**(2), 187–207 (1985). https://doi.org/10.1016/0021-9991(85)90002-6
29. Sanz Leon, P., Knock, S., Woodman, M., Domide, L., Mersmann, J., McIntosh, A., Jirsa, V.: The virtual brain: a simulator of primate brain network dynamics. Frontiers in Neuroinformatics **7** (2013). https://doi.org/10.3389/fninf.2013.00010
30. van Ooyen, A., van Pelt, J.: Activity-dependent neurite outgrowth and neural network development. In: Van Pelt, J., Corner, M., Uylings, H., Lopes Da Silva, F. (eds.) The Self-Organizing Brain: From Growth Cones to Functional Networks, Progress in Brain Research, vol. 102, pp. 245–259. Elsevier (1994). https://doi.org/10.1016/S0079-6123(08)60544-0
31. Van Ooyen, A., Butz-Ostendorf, M.: The rewiring brain: a computational approach to structural plasticity in the adult brain. Academic Press (2017)
32. Yokota, R., Barba, L., Narumi, T., Yasuoka, K.: Petascale turbulence simulation using a highly parallel fast multipole method on gpus. Computer Physics Communications **184**(3), 445–455 (2013). https://doi.org/10.1016/j.cpc.2012.09.011