Computational Neuroscience Breakthroughs Through Innovative Data Management

Farhan Tauheed[†‡], Sadegh Nobari[¶], Laurynas Biveinis[§], Thomas Heinis[‡], and Anastasia Ailamaki[‡]

[†]Data-Intensive Applications and Systems Lab, EPFL, Switzerland
[‡]Brain Mind Institute, EPFL, Switzerland
[¶]National University of Singapore, Singapore
[¶]Department of Computer Science, Aalborg University, Denmark

Abstract. Simulations have become key in many scientific disciplines to better understand natural phenomena. Neuroscientists, for example, build and simulate increasingly fine-grained models (including subcellular details, e.g., neurotransmitter) of the neocortex to understand the mechanisms causing brain diseases and to test new treatments in-silico. The sheer size and, more importantly, the level of detail of their models challenges today's spatial data management techniques. In collaboration with the Blue Brain project (BBP) we develop new approaches that efficiently enable analysis, navigation and discovery in spatial models of the brain. More precisely, we develop an index for the scalable and efficient execution of spatial range queries supporting model building and analysis. Furthermore, we enable navigational access to the brain models, i.e., the execution of of series of range queries where he location of each query depends on the previous ones. To efficiently support navigational access, we develop a method that uses previous query results to prefetch spatial data with high accuracy and therefore speeds up navigation. Finally, to enable discovery based on the range queries, we conceive a novel in-memory spatial join.
The methods we develop considerably outperform the state of the art, but more importantly, they enable the neuroscientists to scale to building, simulating and analyzing massively bigger and more detailed brain models.

1 Introduction

Scientists across many different fields have started to complement their traditional methods for understanding a phenomena in nature with the simulation of spatial models of it. Simulating spatial models has become standard practice in many disciplines and applications. Examples include the simulation of peptide folding [1], star formation in astronomy [2], earthquakes in geology [3], fluid dynamics as well as the brain in neuroscience [4]. To develop a better understanding the scientists continuously increase the size and complexity of the simulations as much as their hardware allows.

Today's tools and algorithms required to cope with the data at the core of simulations, however, cannot cope with the data deluge resulting from bigger and more detailed simulations. While the many spatial indexes [5] developed in the past are of great help to scientists in analyzing and building models, many cannot deal with the complexity and size of today's spatial models.

A particular example of scientists who simulate a detailed model on a massive scale are the neuroscientists of the Blue Brain Project (BBP [4]). In their attempt to understand the brain, i.e., what gives rise to cognition and what mechanisms lead to brain disease, they model and simulate the rat brain (and later the human brain) in great detail. To this end, the neuroscientists build models of the neocortex in unprecedented detail and simulate electrical activity on a supercomputer (BlueGene/P with 16K cores).

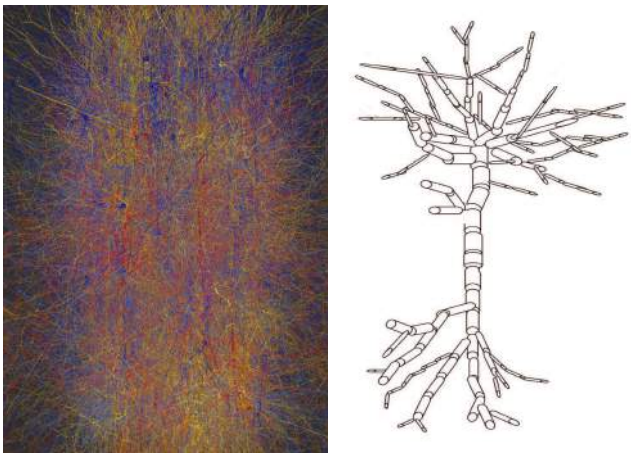


Fig. 1. A visualization of a model microcircuit comprised of thousands of neurons (left) and a schema of a neuron morphology modeled with cylinders (right).

To build detailed models, the neuroscientists in the BBP have analyzed the rat brain tissue in the wet lab over several years and have identified the exact electrophysiological properties and the precise morphological structure of neurons. The neuron morphology defines the branches that extend into large parts of the tissue in order to receive and send out information to other neurons. To obtain a biorealistic model (an example is illustrated in Figure 1, left) the neuroscientists put together thousands or millions of neuron morphologies, each of which is represented by thousands of small cylinders (a morphology is shown in Figure 1, right).

The models built in the BBP have quickly grown in recent years and feature several million neurons today. Although the current model size is still far from

their ultimate goal of building and simulating models as big as the human brain ($\sim 10^{11}$ neurons), today's size already seriously challenges state-of-the-art spatial data management tools.

More important than only the size (number of cylinders) of the models, however, is their level of detail: to build more detailed and biorealistic models the neuroscientists pack more and smaller elements into the same volume. Analyzing the detailed models with current methods is becoming a challenge because the state of the art like the R-Tree [6] do not scale to the increasingly fine-grained/dense models. To address the challenge of increasingly detailed models, we develop a new query execution strategy for dense datasets. At its core is a novel two-phased range query execution strategy where each phase is independent of data density.

The ability to efficiently execute range queries on dense data greatly speeds up building and analyzing of models. Crucial for the further analysis, however, is navigational access to the brain models, i.e., the interactive execution of series of range queries where the location of the next query depends on the result of the previous query. The neuroscientists frequently follow a branch of a neuron and execute spatial range queries for detailed analysis to validate the model. At each location, a query is executed, the data is retrieved as well as visualized before the scientist decides on the next location where a query is executed. Because spatial range query execution is disk bound, executing range query series is a very time-consuming process.

To speed up the execution of interactive range query series, data can be prefetched. State-of-the-art approaches, however, do not prefetch spatial data with high accuracy because they rely on limited information, e.g., previous query positions. We therefore develop a novel approach that prefetches spatial data with a considerably higher accuracy by using the content of previous queries (instead of only their position) and thereby achieve substantially higher prefetch accuracy.

A particular computation that needs to be run based on the query results of each spatial range query and that enables neuroscientific discovery, is placing the synapses in the model. Synapses, i.e., the structures where impulses leap over between neurons, are placed wherever two cylinder of different neurons intersect [7]. Placing synapses therefore is equivalent to an in-memory spatial join where all neurons are tested for intersection.

Given the absence of efficient spatial join methods for memory, we develop a novel in-memory spatial join. The design of our new approach considerably departs from the state of the art and avoids the overlap problem of data-oriented approaches [8, 9] as well as the replication problem of space-oriented approaches [10, 11]. Replication has to be avoided because it (a) increases the memory footprint, (b) requires multiple comparisons and (c) removal of duplicate results. Combining the best of both worlds, our new approach is one order of magnitude faster than known approaches and two orders of magnitude faster than known approaches with a memory footprint of the same size.

In the remainder of this paper we discuss the spatial indexing algorithms we develop in collaboration with the neuroscientists of the Blue brain project to advance computational neuroscience. We describe the three approaches we develop, FLAT [12] for efficient range query execution, SCOUT [13] for the accurate prefetching of spatial data and TOUCH [14] for efficient and scalable in-memory joins, discuss how neuroscientists use them and we demonstrate the considerable impact they have on the process of building the models.

2 Retrieving Dense Neuroscience Data

A crucial type of query in the model building process in the BBP is the spatial range query. Range queries are repeatedly used to visualize parts of the models or to ensure that the models built are biorealistic (testing the tissue density, synapse density or other statistics).

Because today's models of the brain are already very detailed and dense, state-of-the-art indexes to execute range queries [5] are not efficient. The efficient execution of range queries to build and validate models, however, is pivotal today and will become even more important in the future where the models will be increasingly biorealistic and thus dense as the neuroscientists will model phenomena on the subcellular level.

2.1 Motivation

Several spatial access methods supporting the execution of spatial range queries [5] have been developed in the past. While these approaches execute range queries efficiently on many datasets, they unfortunately do not do so on dense or detailed neuroscience models. To make matters worse, they will only scale poorly to more dense and detailed models built in the future.

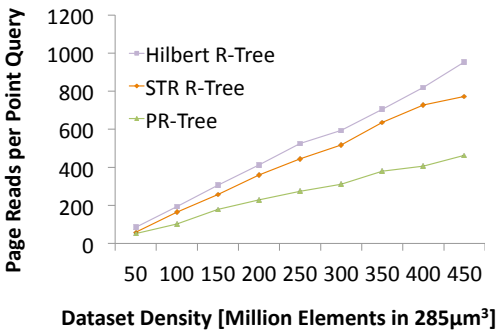


Fig. 2. Point query performance on R-Tree variants.

Current methods are based on the hierarchical organization of the multidimensional spatial data (R-Trees and variants: STR, TGS, PR-Tree, R+-Tree, R*-Tree and others [5]) and therefore suffer from overlap [15] and dead space. Spatially close elements are stored together in the same node (on the same disk page) and a tree is recursively built such that each node has a minimum bounding rectangle (MBR) that encloses all MBRs of its children. Nodes with overlapping

MBRs lead to ambiguity in the tree, i.e., to execute a range query several paths in the tree need to be followed, leading to the retrieval of an excessive number of nodes. To make matters worse, overlap becomes substantially worse with increasing density of the dataset: as the number of spatial elements in the same unit of space increases, so does the overlap of tree-based indexes.

The experiment shown in Figure 2 where we increase the density of the dataset and measure point query performance of different R-tree based approaches [16–18] clearly demonstrates the problem of overlap. Point queries are a good indicator of overlap in R-Trees: the number of nodes retrieved from disk should be in the order of the height of the tree (five in this case) in the absence of overlap. As the results in Figure 2 show, however, the overlap grows rapidly with increasing dataset density, translating into a higher execution time and thus degraded performance.

Despite numerous proposed improvements, e.g., reducing overlap through splitting and replicating elements [15] (thereby increasing the number of nodes in the tree and also its size on disk considerably), the fundamental problem remains the same and needs to be addressed to enable the neuroscientists to build, analyze and validate more detailed and biorealistic models.

2.2 FLAT Query Execution

To enable the neuroscientists in the BBP to build and analyze models of the brain on an unprecedented detailed level, we develop FLAT [12] with a two phased query execution at its core. The key insight we use is that while finding all elements in a particular range query in an R-Tree-like index suffers from overlap, finding an arbitrary element in a range query on the other hand is independent of overlap and therefore is a comparatively cheap operation. Because only one path in the tree has to be followed, the cost of finding an arbitrary element is in the order of the height of the tree.

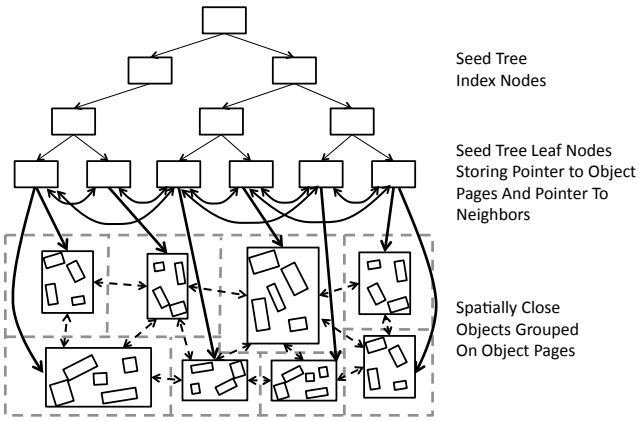


Fig. 3. FLAT: Spatially close elements are packed on the same disk page (rectangle) and pointers (arrows) are added between neighboring pages.

With this insight, we develop a two phased approach where we (1) find an arbitrary element e in the query range and (2) recursively retrieve all other elements (the neighbors of e) within the range. To recursively find the neighbors, FLAT stores

neighborhood information, i.e., what element is next to what other elements. Both phases are independent of overlap and density as the first only depends on the height of the R-tree-like structure and the second only depends on the number of elements in the range query. With both phases avoiding the problem of overlap and only depending on the result size of the query, FLAT will scale to much denser/detailed models.

To limit the amount of information stored, FLAT stores the neighborhood information on the level of groups of elements instead of on the level of single elements, i.e., what spatial element neighbors what other spatial elements. FLAT groups spatially close elements together (and stores them on the same disk page), indexes the groups with an R-Tree index, called the seed tree, and finally computes the neighborhood information between the groups. The neighborhood information itself is stored in the leaf nodes of the R-Tree (seed tree).

Figure 3 illustrates how queries are executed using the groups of elements and the neighborhood information: first FLAT retrieves an arbitrary group in the query range and then recursively retrieves all neighboring elements in the query range.

While some datasets may contain inherent neighborhood information (e.g., in meshes the neighborhood is stored in the edges), many (and in particular the neuroscience datasets) do not have any or only limited neighborhood information. As a consequence, to make FLAT work on arbitrary datasets, we add neighborhood information to the index in a preprocessing phase. Computing and storing neighborhood information is not an undue burden as the increase in index building time is below 10% and the space increases by only 12% (both compared to the STR [16] bulkloading approach).

2.3 Impact of FLAT

The impact FLAT on the work of the neuroscientists is considerable. Until recently they have not been able to build, analyze and validate models exceeding one million neurons. With FLAT they now have the ability to scale to bigger and, more importantly, to much more detailed models. An experiment with a neuroscience dataset shows the promise of FLAT. In this experiment we execute 200 small sized queries (each with a volume of $5 \times 10^{-7}\%$) used for structural analysis on a dataset where the density increases from 50 to 450 million elements in a constant volume of $285 \mu m^3$. We increase the number of elements in the same volume to emulate increasingly detailed models.

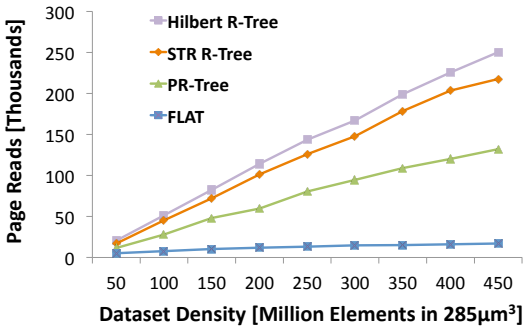


Fig. 4. Execution time for executing 200 the small structural analysis queries of size $5 \times 10^{-7}\%$

As Figure 4 impressively demonstrates, FLAT already today considerably outperforms R-Tree based approaches by a factor of 8 for the densest model (with 450 million elements). More importantly, however, the trend clearly shows that FLAT will scale better to more detailed and dense models in the future. This is pivotal as it finally enables the neuroscientists to build more bio-realistic models where subcellular elements (e.g., neurotransmitters) can be precisely replicated for a more accurate simulation of brain activity.

3 Prefetching for Structure Following Spatial Queries

FLAT enables the neuroscientists to efficiently execute spatial range queries on todays models and also on future, even more dense & detailed spatial models. More important for many of their analysis, however, is it to execute a series of range queries: following a structure in the model, e.g., a neuron branch, they need to execute several range queries to assess the quality or validity of the model. On the result of each range query they compute different types of statistics (tissue density, synapse placement, synapse count, etc). Series of range queries are not only crucial for the neuroscientists, but also for other scientists who analyze road networks, arterial trees and others.

Executing a series of range queries is an interactive process where the user follows a structure, executes a query, computes one or several statistics, analyzes the statistics and then decides on the location of the next query and executes it. Because the series is interactive, the disk is idle during the computation of statistics (between two range queries) and data can be prefetched to speed up the series. State-of-the-art approaches, however, rely on limited information to predict the next query location and thus prefetch with low accuracy.

3.1 Motivation

Known approaches used to prefetch spatial data do not have good enough accuracy as they only use limited information of previous queries to predict the location of the next query. Several state-of-the-art approaches rely on the positions of past queries. One particular approach [19] uses the last query position and prefetches around it. More sophisticated approaches [20] use the last few positions, fit a polynomial into them and extrapolate the polynomial to predict the next query location. Series of range queries on neuron structures, however, are very jagged and not smooth at all. The irregular structure makes it very hard to interpolate accurately with a polynomial and consequently this class of prediction approaches does not prefetch with good accuracy.

Another class of approaches [21] attempts to learn from past user behavior by keeping track of all paths visited in the past. Prefetching, i.e., predicting the next query location, is based on the history. Because the models in our scenario are so massive, it is unlikely that any path will be visited twice, therefore making prefetching strategies based on past paths visited inaccurate.

3.2 Content-Aware Prefetching

To prefetch more accurately and to considerably speed up the execution of series of range queries and therefore analysis, we develop SCOUT [13]. SCOUT departs form previous approaches as it does not only consider previous query positions, but also takes into account the previous query content, knowing that the scientist follows one of the structures in the previous queries. As a consequence, SCOUT prefetches with a considerably higher accuracy, speeding up query series by a factor of up to 15×.

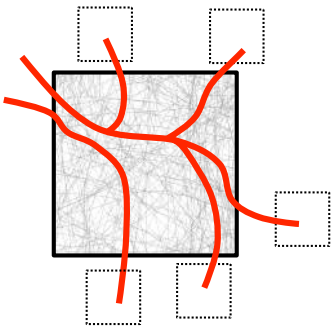


Fig. 5. Prefetching of spatial data at the exit locations of the structure.

SCOUT summarizes the content of the most recent query q, i.e., it identifies the topological skeleton in q and approximates it with a graph. The graph of q represents all the structures the neuroscientist is potentially following and SCOUT therefore prefetches data at all locations where the graph leaves q (exit locations). Range queries are executed to prefetch data at these locations until the user executes a new query in the series. Figure 5 shows how small range queries are executed at the exit locations of the last query.

As the example in Figure 5 shows, in some cases SCOUT has to prefetch in multiple locations. This is the case at the beginning of a series of queries where SCOUT cannot yet identify the one structure the neuroscientist follows. By using iterative candidate pruning, however, SCOUT can reliably identify the neuron branch followed after a few queries already.

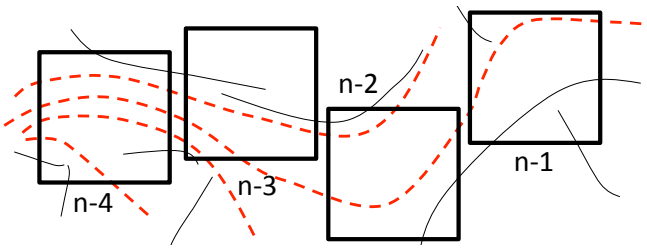


Fig. 6. Pruning the irrelevant structures (solid lines) from the candidate set (dashed lines) in subsequent queries (solid squares) of the series.

Iterative candidate pruning exploits that all previous queries must contain the branch the scientist follows. To prefetch for the n^{th} query, SCOUT thus only needs to consider the set of branches leaving the $(n-2)^{th}$ query and the set of branches entering the $n-1^{th}$ (most recent) query. The branch followed is in

the intersection of both sets. As the number of queries in a series increases, the number of branches in the intersection between two consecutive queries decreases continuously and the branch the user follows can be identified reliably. Figure 6 illustrates how through iteratively reducing the set of candidates, SCOUT can reliably identify the structure the scientist follows after only a few queries.

3.3 Impact of SCOUT

SCOUT helps the neuroscientists to make building and analyzing models substantially faster. The speedup for different types of analysis ranges between 4× and 15× and enables a significantly faster turnaround from model building to analysis.

The experiment in Figure 7 shows this impressively by comparing the cache hit rate and the speedup with state-of-the-art approaches. In this experiment we have executed different series of range queries from different applications (visualization, model building) which differ in size of each query (30'000 to 80'000μm^3) as well as the length of the series (ranging from 25 to 65).

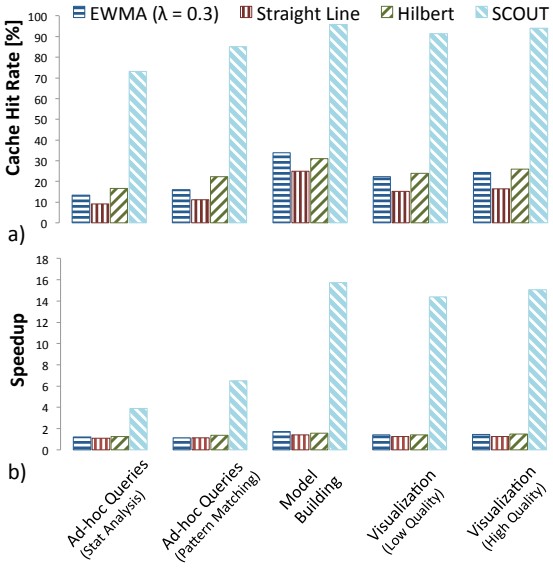


Fig. 7. Accuracy of the approaches for all microbenchmarks (a) and speedup of the approaches for all microbenchmarks (b).

The experiment shows that SCOUT speeds up the analysis considerably: the cache hit rate of SCOUT is 4× higher for the visualization query series resulting in a 7× higher speedup compared to EWMA [22], the fastest state-of-the-art approach (based on polynomial extrapolation). Like FLAT, SCOUT also speeds up the model building process and allows the neuroscientists to build bigger models considerably faster.

4 In-Memory Spatial Join for Model Building

A particular computation the neuroscientists need to execute on the result of each of the queries in a series is the *touch detection*. In this computation the neuroscientists determine where to place synapses, the structure thats permit an electrical impulse to leap over between neurons, in the model. Experiments in the wet lab have shown that it suffices to place synapses where branches of neurons

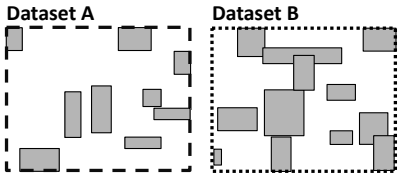
intersect [7] to obtain a biorealisitc model of the brain. Touch detection therefore needs to find cylinders (compare with Figure 1, left) of different neurons that overlap/intersect with each other, translating this process into an in-memory spatial join. While many spatial join methods have been developed for disk in recent years, no scalable in-memory approach exists.

4.1 Challenge

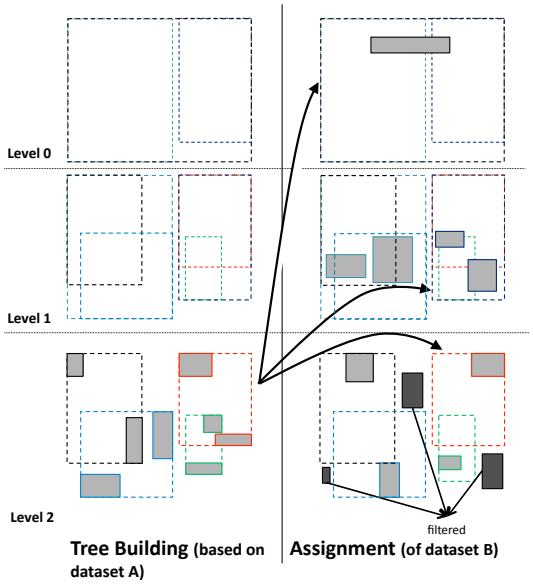
For the touch detection, the neuroscientists need to execute a spatial join on every result of the query series. Because the result of each query fits into the memory of even desktop machines, the spatial join needs to be performed in-memory.

Despite decades of research into spatial joins, only two algorithms have been developed to join two datasets in memory: the nested loop join [23] and the sweep line approach [24]. Neither of the two scales well: the nested loop join has a complexity of $O(n^2)$ whereas the sweep line approach becomes inefficient when too many elements are on the sweep line (very likely in case of dense data/detailed neuroscience models).

Approaches primarily developed for disk [25] can of course also be used in memory. Existing work can be categorized into space- or disk-oriented partitioning approaches. Besides advantages, both classes also have clear disadvantages: space-oriented approaches [10, 11] generally need to replicate elements (elements that intersect with two partitions are copied to both) leading to considerable overhead and multiple detection of the same intersections whereas data-oriented approaches [8] suffer from the overlap problem of R-Trees which degrades performance considerably, particularly when used with dense datasets.



(a) The datasets A and B



(b) Tree building, assignment and joining phases

Fig. 8. The three phases of : building the tree, assignment and joining.

4.2 TOUCH: Efficient In-Memory Spatial Join

Given the lack of in-memory spatial join approaches as well as the challenges of either data- and space oriented approaches, we have developed TOUCH [14], a novel in-memory spatial join algorithm. With TOUCH we want to avoid space-oriented partitioning because it typically leads to replication of elements. Replication has to be avoided because it (a) increases the memory footprint and (b) requires multiple comparisons between copies of elements (as well as making the removal of duplicate results necessary). Data-oriented partitioning on the other hand has the problem of overlap resulting in degraded performance, particularly on dense datasets.

With TOUCH we want to combine the best of both, space- as well as data-oriented partitioning, while avoiding the pitfalls. We use data-oriented partitioning to avoid the replication problem of space-oriented partitioning and build an index based on data-oriented partitioning (similar to an R-Tree) on the first dataset A (all elements of A are in the leaf nodes). To avoid the issue of overlap, we do not probe the data-oriented index for every element of the second dataset B. Instead, we assign each element b of B to the lowest (closest to the leafs) internal node of the index that fully contains b. Once all elements of B are assigned to the R-Tree, they are joined: the elements of B in a particular internal node n are joined with with all leaf nodes (containing elements of A) reachable from n. Figure 8 shows the process, i.e., how an index is built based on dataset A, how the elements of dataset B are assigned to internal nodes and finally, how internal nodes are joined with leaf nodes.

We further improve TOUCH's performance by using the *filtering* concept from space-oriented partitioning. When indexing dataset A, i.e, building an index on A, some space may not be covered by the leaf nodes. Consequently, if any element of B falls into this empty space, it cannot intersect with any element of A and thus does not have to be considered. Depending on the distribution of the dataset, filtering can considerably reduce the number intersection tests between elements.

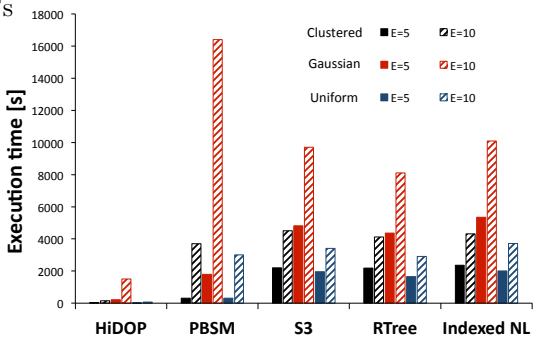


Fig. 9. Comparing the approaches for two different distance predicates ϵ on all datasets.

4.3 Impact of TOUCH

TOUCH is one order of magnitude faster than known approaches and two orders of magnitude faster than known approaches with an equally small memory footprint. An experiment with neuroscience data shows this in Figure 9: in this

experiment we measure how fast the spatial join can be computed on datasets of size 50 millions with disk-based join methods used in memory for two different distance predicates (we exclude the two in-memory join methods because they are too slow). We use three different datasets where the locations of the objects are distributed uniform, gaussian and clustered (100 clusters, each with a gaussian distribution). Clearly TOUCH is the fastest for either distance predicate, followed by PBSM [11]. Although PBSM is the fastest competitor, it is still one order of magnitude slower and uses considerable more memory (a factor of 8 $\times$ more).

TOUCH makes a considerable difference in the work of the neuroscientists. Without TOUCH, the biggest model they touch detected was 1 million neurons big. Today with TOUCH, they are able to touch detect a model of 10 millions and they are working on touch detecting a 33 million neuron model. Although TOUCH is very conservative with respect to memory, running touch detection on bigger models is not possible because a 33 million neuron model entirely fills the memory of their current infrastructure. We are currently investigating out of core methods where disk capacity is the bottleneck and no longer memory.

5 Conclusions

To gain a better understanding of how the brain works, what gives rise to cognition and what mechanisms govern dementia, the researchers of the Blue Brain Project build increasingly big, complex and detailed models of the brain. Because their models are so big and detailed, state-of-the-art methods no longer can be used to efficiently build, analyze and validate them. We have therefore developed FLAT, a method for the scalable execution of spatial range queries on dense spatial models, TOUCH, a prefetching method for spatial data used to speed up series of spatial range queries and finally TOUCH, an efficient in-memory spatial join approach.

The impact of the methods we have developed on the Blue Brain project are substantial: the neuroscientists can build bigger and more detailed models faster. The limit of models they can build has grown considerably to 33 million neurons today. Dealing with increasingly detailed and complex spatial models is not just a problem of neuroscientists, but is shared in many scientific disciplines that simulate natural phenomena and the algorithms we developed thus have impact beyond neuroscience.

Our results also demonstrate that despite decades of research in spatial data management, many problems still remain open. Increasing main memory as well as novel storage technology (in the memory hierarchy), for example, means that several spatial indexes need to be redesigned as we have shown with TOUCH. The execution of spatial range queries or nearest neighbor queries, for example, needs to be supported with radically different indexes optimized for memory.

New types of datasets (e.g., dense, complex spatial datasets) make new indexes necessary (FLAT) and new types of queries (e.g., series of range queries) also call for the development of new indexes. Nearest neighbor queries with con-

straints (find the nearest neuron with a given voltage), for example, are not yet supported efficiently.

Finally, also the massive scale of the brain models as well as the size of the simulation results mandates new methods for data management. Analyzing the models with spatial queries as well as the simulation output with spatio-temporal queries to find interesting phenomena needs to be supported with scalable methods. Large-scale parallel approaches to analyze massive spatial or spatio-temporal therefore will have to be developed.

References

1. Gnanakaran, S., Nymeyer, H., Portman, J., Sanbonmatsu, K.Y., Garcia, A.E.: Peptide folding simulations. Current Opinion in Structural Biology **13**(2) (2003) 168–174
2. Gray, J., Szalay, A., Thakar, A., Kunszt, P., Stoughton, C., Slutz, D., Vandenberg, J.: Data Mining the SDSS SkyServer Database. In: Technical Report, MSR-TR-2002-01, Microsoft Research. (2002)
3. Komatitsch, D., Tsuboi, S., Ji, C., Tromp, J.: A 14.6 Billion Degrees of Freedom, 5 Teraflops, 2.5 Terabyte Earthquake Simulation on the Earth Simulator. In: International Conference on Supercomputing (SC '03)
4. Markram, H.: The Blue Brain Project. Nature Reviews Neuroscience **7**(2) (2006) 153–160
5. Gaede, V., Guenther, O.: Multidimensional Access Methods. ACM Computing Surveys **30**(2) (1998)
6. Guttman, A.: R-trees: a Dynamic Index Structure for Spatial Searching. SIGMOD '84
7. Kozloski, J., Sfyrakis, K., Hill, S., Schürmann, F., Peck, C., Markram, H.: Identifying, Tabulating, and Analyzing Contacts Between Branched Neuron Morphologies. IBM Journal of Research and Development **52**(1/2) (2008) 43–55
8. Brinkhoff, T., Kriegel, H., Seeger, B.: Efficient Processing of Spatial Joins R-Trees. SIGMOD '93
9. Lo, M.L., Ravishankar, C.V.: Spatial Joins Using Seeded Trees. In: SIGMOD '94
10. Koudas, N., Sevcik, K.C.: Size Separation Spatial Join. In: SIGMOD '97
11. Patel, J.M., DeWitt, D.J.: Partition Based Spatial-Merge Join. In: SIGMOD '96
12. Tauheed, F., Biveinis, L., Heinis, T., Schürmann, F., Markram, H., Ailamaki, A.: Accelerating range queries for brain simulations. In: ICDE '12
13. Tauheed, F., Heinis, T., Schürmann, F., Markram, H., Ailamaki, A.: Scout: Prefetching for latent structure following queries. In: VLDB '12
14. Nobari, S., Tauheed, F., Heinis, T., Karras, P., Bressan, S., Ailamaki, A.: TOUCH: In-Memory Spatial Join by Hierarchical Data-Oriented Partitioning. In: SIGMOD '13
15. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. VLDB '87
16. Leutenegger, S.T., Lopez, M.A., Edgington, J.: STR: a Simple and Efficient Algorithm for R-tree Packing. ICDE '97
17. Arge, L., Berg, M.D., Haverkort, H., Yi, K.: The Priority R-tree: A Practically Efficient and Worst-case Optimal R-Tree. ACM Transactions on Algorithms **4**(1) (2008) 1–30

18. Kamel, I., Faloutsos, C.: Hilbert R-Tree: An Improved R-Tree using Fractals. In: VLDB '94
19. Park, D.J., Kim, H.J.: Prefetch policies for large objects in a web-enabled gis application. Data Knowledge Engineering **37**(1) (2001) 65–84
20. Chan, A., Lau, R.W.H., Si, A.: A motion prediction method for mouse-based navigation. In: International Conference on Computer Graphics. 139–146
21. Lee, D., Kim, J., Kim, S., Kim, K., Yoo-Sung, K., Park, J. In: Adaptation of a Neighbor Selection Markov Chain for Prefetching Tiled Web GIS Data. Volume 2457 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2002) 213–222
22. Chim, J., et al.: On caching and prefetching of virtual objects in distributed virtual environments. In: MULTIMEDIA '98
23. Mishra, P., Eich, M.H.: Join Processing in Relational Databases. ACM Computing Surveys **24**(1) (1992) 63–113
24. Edelsbrunner, H.: Algorithms in combinatorial geometry. Springer-Verlag (1987)
25. Jacox, E.H., Samet, H.: Spatial join techniques. ACM TODS **32**(1) (2007) 7