Enabling Scientific Discovery Via Innovative Spatial Data Management

Thomas Heinis, Farhan Tauheed, Mirjana Pavlovic, Anastasia Ailamaki
Data-Intensive Applications and Systems Laboratory
École Polytechnique Fédérale de Lausanne, Switzerland
{firstname.lastname}@epfl.ch

Abstract

Researchers in several scientific disciplines are struggling to cope with the masses of data resulting from either increasingly precise instruments or from simulation runs on ever more powerful supercomputers. Efficiently managing this deluge of data has become key to understand the phenomena they are studying. Scientists in the simulation sciences, for example, build increasingly big and detailed models, as detailed as the hardware allows, but they lack the efficient technology to update and analyze them.

In this paper we discuss how innovative data management techniques we have developed, enable scientists to build and analyze bigger and more detailed spatial models and how these techniques ultimately accelerate discovery in the simulation sciences. These include spatial join methods (in memory and on disk), techniques for the efficient navigation in detailed meshes, an index for range query execution on complex and detailed spatial data as well as in-memory mesh indexes. The evaluation of these techniques using real neuroscience datasets shows a considerable performance improvement over the state of the art, and that the indexes we proposed scale substantially better for the purpose of the analysis of bigger and denser spatial models.

1 Introduction

The simulation of spatial models has become a standard practice complementing traditional methods for understanding natural phenomena across many scientific disciplines. Examples cover various domains and include the simulation of peptide folding [3], star formation in astronomy [4], earthquakes in geology [6], fluid dynamics as well as the brain simulation in neuroscience [9].

Although the scientific disciplines are vastly different, the process of building and simulating a model is identical. As Figure 1 illustrates, data from the wet lab, from literature and from medical records is used to assemble an initial model. The model is then analyzed, validated and finally simulated. During and also after simulation, the model needs to be analyzed and visualized. The results of the visualization and analysis are finally fed back to the building phase to build more realistic models. The unprecedented amounts of data in every phase throughout the building and simulating process makes the data management in simulations challenging.

It is, however, not only the amount of data that keeps on growing and challenging current indexing methods. To develop a better understanding the scientists continuously increase the size and complexity of the simulations,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

pushing the limits of their computing infrastructure. In neuroscience, for example, scientists build models on the subcellular level (e.g., modelling neurotansmitter), thereby making the model considerably more detailed and dense. The data management challenges pertain to the size as well as complexity or level of detail of the models.

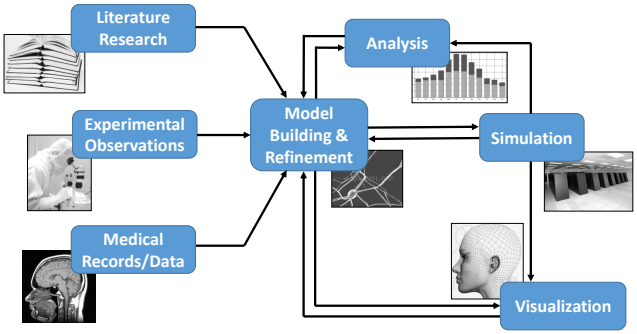


Figure 1: Steps of a simulation workflow.

The increasing size and complexity, i.e., the density (number of spatial elements per volume), of spatial datasets used in simulations renders existing tools and algorithms inadequate. Increasing density (or level of detail) in simulation datasets challenges the standard indexing approaches like the R-Tree (or its improvements [2]). The more elements a spatial dataset packs in the same space, i.e., the denser it is, the more the elements on the leaf level of the R-Tree overlap. More detrimental to query execution time, however, is that more overlap on the leaf level translates into disproportionately more overlap in the tree structure of R-Tree [15] and query execution time of the R-Tree consequently does not scale well with density. Similarly, the update frequency and scale in simulation applications (for example in earthquake or computational fluid dynamics simulations almost all elements change position in every step) render known update mechanisms for spatial indexes inefficient. Novel indexing techniques consequently need to be developed to provide efficient and scalable access to dense and frequently updated spatial datasets.

In this paper we identify the state-of-the-art spatial indexes used in the simulation workflow that do not scale with increasing density/level of detail of the spatial models. We develop new indexes for range query execution, spatial join and prefetching of spatial data based on the idea that while density prevents the state of the art from scaling, the proximity of the elements in dense datasets can be used to our advantage. More particularly, the approaches we develop are based on the key insight that the connectivity (inherent or added) in datasets can be used to recursively explore datasets independent of density (and hence avoiding the overlap problem).

In the remainder of this paper we present the data management techniques based on connectivity to support building and simulating large-scale models. The paper is organized along the simulation workflow: we first give an overview over the methods we have developed for building [11, 14] models, for analyzing [15, 17] spatial models and finally for analyzing running simulations [12, 16]. We subsequently conclude and discuss the impact the connectivity-based indexing techniques have on a neuroscience use case.

2 Model Analysis & Visualization

A crucial type of query in the model building process and in the model analysis phase is the spatial range query. Range queries are repeatedly used to visualize parts of the models or to ensure that the models built satisfy statistical constraints (in case of neuroscience, for example, testing the tissue density, synapse density, etc.). It is also equally important for many of their analysis to execute a series of range queries that follow one of the many structures (e.g., a road in a road network in GIS) in the model to assess the quality or validity of the model.

2.1 Range Queries on Dense Spatial Models

The efficient execution of range queries to build and validate models is pivotal today and will become even more important in the future where the models will be increasingly dense (i.e., biorealistic through modeling phenomena on the subcellular level). While today's spatial indexing approaches [2] execute range queries efficiently on many datasets, they unfortunately do not do so on today's detailed and dense spatial models [15]. To make matters worse, they will only scale poorly to more detailed future models as experiments show [15].

Today's methods are based on the hierarchical organization of spatial data (R-Trees and variants [2]) and therefore suffer from overlap [2] and dead space. Overlap increases considerably with increasing level of detail/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 [15]. Despite numerous proposed improvements, e.g., reducing overlap through splitting and replicating elements (R+-Tree [2]), the fundamental problem remains the same and needs to be addressed to enable the simulation scientists to build and analyze more detailed models.

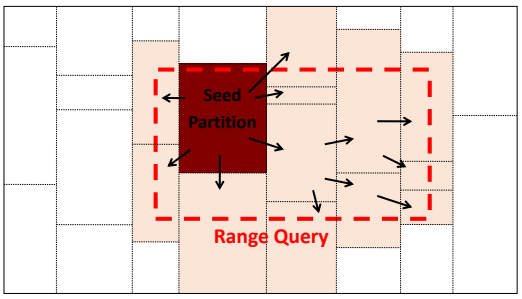


Figure 2: Query execution: given an inital group of elements (dark), FLAT recursively visits all neighbors of the seed partition.

To enable scientists to build and analyze models with an unprecedented level of detail, we developed FLAT [15]. The key insight we use is that while finding all elements in a particular range query in an R-Tree suffers from overlap, finding an arbitrary element in a range query is independent of overlap and thus is a cheap operation. With this insight, we develop a two-phase query execution approach where we (1) find an arbitrary element e in the query range using an R-Tree and (2) recursively retrieve all other elements (the neighbors of e) within the range using neighborhood information (what elements neighbor what other elements) previously added to the dataset. Both phases are independent of overlap and density as the first only depends on the height of the R-tree and the second only depends on the number of elements in the range query. FLAT thus becomes independent of overlap and scales to much denser/detailed models.

When indexing, FLAT needs to build an R-Tree used to find an initial element and it also needs to compute neighborhood information of the elements. To curb the amount of information stored, FLAT only stores neighborhood information on the level of groups of elements instead of on the level of single elements. FLAT groups spatially close elements together (and stores them on the same disk page), indexes the groups with the R-Tree and finally computes the neighborhood information between the groups. The neighborhood information itself is stored in the leaf nodes of the R-Tree. Figure 2 illustrates how queries are executed using the groups of elements: given an initial arbitrary group in the query range, FLAT recursively retrieves all neighbors until all groups in the query range are retrieved.

FLAT works on arbitrary datasets from different disciplines and as measurements show it can speed up query execution by up to one order of magnitude on a dense spatial model from neuroscience [15]. The improvement over the state of the art is bigger, the more densely packed the spatial datasets are. Indeed, similar results are obtained for equally densely packed spatial datasets from astronomy and computer vision [15].

2.2 Executing Guided Spatial Range Query Sequences

An important type of query for the model analysis process is the execution of a series of range queries: following a structure in the model, e.g., a neuron branch, neuroscientists need to interactively execute range queries to analyze 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 sequence of range queries is an interactive process where a scientist follows a structure, executes a query, performs analyses or computes statistics on the query result, studies the analyses results and then decides on the location of the next query and executes it. The disk is idle during the computation of statistics (between two range queries) because execution of the series is interactive and the next query location is unknown. To speed up the execution of the series data can be prefetched at potential query locations while the disk is otherwise idle.

State-of-the-art approaches to prefetch spatial data only do so with low accuracy because they rely only on limited information, i.e., the position, of previous queries to predict the location of the next query. One particular

approach [13] uses the last query position and prefetches around it. More sophisticated approaches [1] use the last few positions, fit a polynomial and extrapolate the polynomial to predict the next query location. Series of range queries on structures like neurons or arterial trees, however, are not smooth at all but jagged and are therefore impossible to interpolate accurately with a polynomial. A different class of approaches [8] learns from past user behavior by keeping track of all paths visited in the past and by basing predictions on the accumulated history. Given the massive size of today's spatial models, however, it is unlikely that any path will be visited twice, therefore making prefetching strategies based on past paths visited inaccurate.

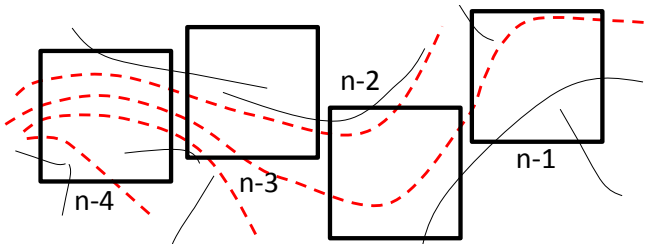


Figure 3: Pruning the irrelevant structures (solid lines) from the candidate set (dashed lines) in subsequent queries (solid squares) of the series.

To prefetch more accurately and to speed up the execution of series of range queries we develop SCOUT [17]. SCOUT departs from previous approaches because it does not consider previous query positions, but instead takes into account the previous query content (the structures in the previous query) knowing that scientists are likely to follow one of the structures in the previous queries.

SCOUT summarizes the structures 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 scientist is potentially following and SCOUT therefore prefetches data at all locations where the graph edges and therefore the structures exit query q. Range queries are executed to prefetch data at these locations until the user executes a new query in the series.

Clearly, having to prefetch at several different locations will reduce accuracy as only one of the prefetched locations is correct. SCOUT uses candidate pruning to reduce the number of prefetching locations, exploiting 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. Figure 3 shows a series of range queries ($n-1$ to $n-4$th query) obtained by a neuroscientists interactively executing queries, i.e., through executing $n-4$ and depending on the result executing query $n-3$. The figure also illustrates how through iteratively reducing the set of candidates, SCOUT can reliably identify the structure the scientist follows after only a few queries, ultimately speeding up query series by a factor of up to 15× [17].

3 Model Building

A crucial operation in building spatial models is to find the intersections of spatial elements. In many simulation models, overlap or intersection between elements no longer realistically reflects the system modeled (neurons cannot overlap in reality and neither can celestial objects). A spatial join is consequently needed in the model building phase to detect errors, i.e., intersections.

Neuroscientists use the spatial join for yet a different application: to obtain a biorealistic model they need to determine where to place synapses (structures that permit electrical impulses to leap between neurons). Prior research shows [7] that synapses need to be placed where neurons are within a given distance to each other, translating the problem into a spatial or distance join.

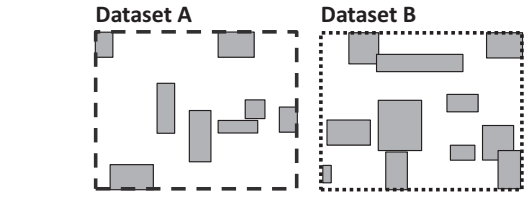
3.1 In-Memory Spatial Join

If the spatial model fits into the main memory of a single machine or in the aggregate memory of a supercomputer, the spatial join needs to be performed in memory. Despite decades of research into spatial joins, only two algorithms have been particularly designed to join two datasets in memory: the nested loop join [5] and the sweep line approach [5]. Neither of the approaches scales well: the nested loop join has quadratic complexity while the sweep line approach is inefficient in case too many elements are on the sweep line (excessive comparisons of elements nearby in one dimension but distant in a different dimension).

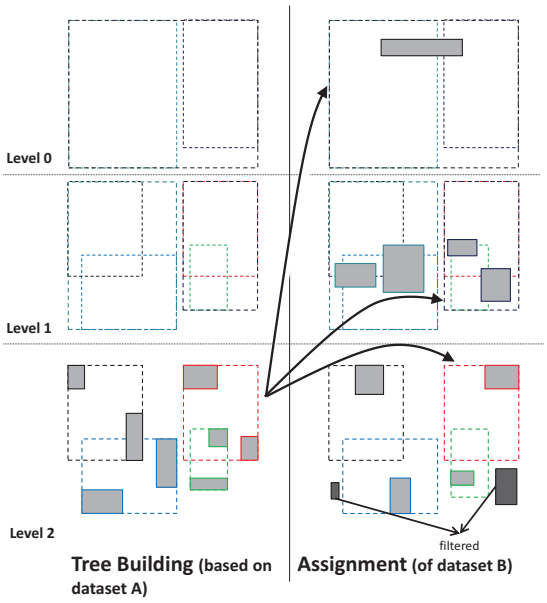
Existing work developed for disk [5] can of course also be used in memory. Disk-based spatial joins can be categorized into space- or disk-oriented partitioning approaches and while both classes have advantages, they also have clear disadvantages: space-oriented approaches [5] need to replicate elements (elements intersecting with two partitions need to be copied to both) increasing the memory footprint and incurring multiple detections of the same intersections. Data-oriented approaches [5], on the other hand, suffer from the overlap problem shared by all R-Trees. Overlap in data-oriented approaches degrades performance already today, but more importantly, it will increase with denser future datasets [15].

Given the shortcomings of current in-memory spatial join approaches and the challenges of disk-based spatial joins, we develop a novel in-memory spatial join algorithm called TOUCH [11]. TOUCH avoids space-oriented partitioning because space-oriented partitioning requires replication of elements which (a) increases the memory footprint and (b) requires multiple comparisons between copies of elements (as well as making the removal of duplicate results necessary). TOUCH also targets at avoiding the problem of overlap prevalent in approaches based on data-oriented partitioning.

TOUCH uses data-oriented partitioning to avoid the replication problem and builds an index 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, it does not probe the index for each element of the second dataset B. Rather, it assigns each element b of B to the lowest (closest to the leafs) internal node of the index that fully contains b. Only after all elements of B are assigned to nodes of the index, TOUCH performs the actual join: elements of B in each internal node n are tested for intersection with all leaf nodes (containing elements of A) reachable from n. Figure 4 illustrates the process, i.e., how the index is built on dataset A, how the elements of dataset B are assigned to nodes and finally, how internal nodes are joined with leaf nodes.



(a) The datasets A and B



(b) Tree building, assignment and joining phases

Figure 4: The three phases of TOUCH: building the tree, assignment and joining.

Our measurements show that TOUCH outperforms known in-memory approaches as well as disk-based approaches used in memory [11]. TOUCH is the fastest to perform the join followed by PBSM [5], a simple space-partitioning approach used in memory. Although PBSM is the fastest competitor, it is still one order of magnitude slower and also uses considerable more memory (a factor of 8 $\times$ more).

3.2 Selective On-Disk Spatial Join

The spatial join, however, is not only crucial in memory. To build large-scale models that do not fit into main memory, efficient out-of-core methods are required to support simulation scientists. A particularly important disk-based spatial join needed by simulation scientists is the joining of datasets of different density, i.e., of similar spatial extent but with a vastly different number of spatial elements. Example applications include adding a few roads or other spatial objects to GIS datasets, adding the branches of one neuron to a spatial model of the neocortex and similar applications. The efficiency of the join is pivotal as it is oftentimes executed repeatedly to join several sparse datasets with one dense dataset.

Many approaches for disk-based spatial joins [5] have been developed in the past and each can be used to join a dataset A_i (with few elements) and B (with a massive number of elements). Existing approaches, however, are not efficient for a join where with a very small A_i, only a small subset of B needs to be retrieved (and tested against A_i). State-of-the-art approaches based on space-oriented partitioning (e.g., PBSM [5]) create coarse-grained partitions and thus the entire dataset B is read for a join, leading to excessive disk access. Approaches based on data-oriented partitioning, on the other hand, require hierarchical trees (e.g., synchronized R-Tree [5]) to access the data and thus suffer from the well documented problems of overlap, also resulting in excessive disk access.

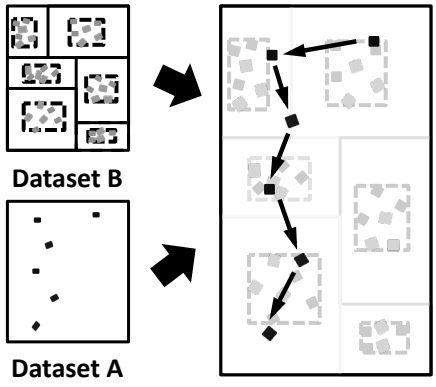


Figure 5: GIPSY uses the sparse dataset to walk/crawl through the dense dataset.

We develop GIPSY, a novel approach that avoids the coarse-grained partitioning of space-oriented approaches and instead uses the fine-grained data-oriented partitioning, thereby enabling the join to read from B only the small subset required. At the same time GIPSY avoids the excessive disk page reads and comparisons due to overlap in the tree structure of data-oriented approaches. Instead of traversing a tree like data-oriented approaches (e.g., the R-Tree), GIPSY traverses the data using a crawling approach [15].

More precisely, GIPSY partitions dataset B in data-oriented fashion and adds neighborhood information to B, i.e., what elements neighbor what other elements. It then takes the elements of the sparse dataset A_i and visits them one after the other by walking between them using the neighborhood information previously added to the dense dataset B. Once GIPSY arrives at the location of a particular element e_i of the sparse dataset, it uses crawling (again using the neighborhood information) to find all elements of the dense dataset around e_i's location and tests them for intersection with e_i. Once no more elements intersecting with e_i can be found, GIPSY walks to the position of the next spatial element e_{i+1} of A_i in the dense dataset B. Figure 5 illustrates how GIPSY uses the sparse dataset to direct walking in the dense dataset.

With its novel combination of crawling and data-oriented partitioning, GIPSY achieves a 2 to 18$\times$ speedup compared to the fastest approaches (indexed nested loop and PBSM, both in [5]) when joining several A_i with B. The evaluation [14] further shows that the improvement over the state of the art grows considerably bigger, the more sparse datasets A_i are joined with B or the bigger the difference in density between A_i and B is.

4 Model Simulation

To analyze, steer and monitor the spatial model while the simulation runs, a number of spatial range queries needs to be executed on the spatial model entirely stored in main memory of the simulation infrastructure at every step of the simulation. Indexing the spatial model speeds up range query execution, however, current indexing methods cannot efficiently support the large-scale updates of simulation applications. During simulation the

spatial models undergo massive changes, resulting in a change of position of all spatial elements in the model at every step of the simulation. Maintaining an index in face of changes on this scale incurs considerable overhead.

Current methods particularly designed to support large-scale updates on spatial indexes cannot cope with simulation applications: not enough queries are executed on the index at every time step to amortize the cost of rebuilding lightweight spatial indexes (MOVIES [10]) at every time step or the cost maintaining indexes designed specifically to reduce update cost (LUR-Tree or QU-Trade [10]). Approaches particularly designed for moving objects that index trajectories (e.g., STRIPES, TPR-Tree [10]) cannot be used either because the unpredictable nature of the movements in simulation applications cannot be interpolated with polynomials.

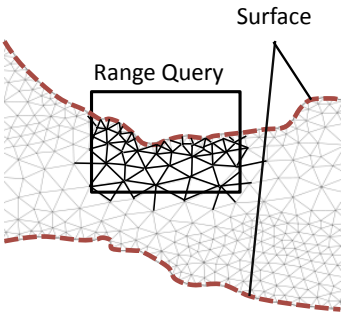


Figure 6: OCTOPUS: Starting from the surface, the edges of the polyhedra will be visited.

To support the efficient execution of range queries on entirely main memory resident meshes undergoing frequent and massive changes we develop OCTOPUS [16]. Even with unpredictable changes affecting the entire mesh dataset OCTOPUS outperforms state-of-the-art methods by working on the mesh datasets directly instead of maintaining complex data structures (like indexes). OCTOPUS uses the connectivity of the mesh to avoid accessing the entire mesh when computing query results: given any vertex inside the query region, OCTOPUS can use the mesh connectivity to recursively retrieve all vertices inside the query. Using the current state of the mesh directly in memory has the advantage that the result can be computed without having to consider the change of the vertex location in the last update.

Solely depending on the connectivity of the mesh, however, bears the risk that the result is incomplete because parts of the mesh in the query region may not be connected (as illustrated in Figure 6). OCTOPUS therefore starts the mesh traversal from the mesh surface enclosed in the query region and recursively retrieves all neighboring elements within the query range (in the absence of a surface vertex in the query range, it starts a directed walk from a random element to find a vertex inside the range). By building on the key insight that every vertex inside a query range is connected to at least one vertex on the surface, we prove [16] that OCTOPUS retrieves the complete result. To find start elements on the mesh surface, OCTOPUS maintains a set of pointers to the surface elements. The set only infrequently needs maintenance because the surface only changes in the rare event where the connectivity of the mesh changes.

Our experiments show that OCTOPUS achieves a speedup between 7.2 and 9.2$\times$ compared to the state of the art. OCTOPUS will scale better with increasingly detailed meshes than other approaches because it only needs to keep track of the mesh surface: when meshes become more detailed, the size of surface grows only quadratic whereas the size of the complete mesh grows cubic. We also develop a general version for the efficient execution of spatial range queries on arbitrary data other than meshes [12].

5 Conclusions

To study in more detail how a natural phenomena works, scientists in different disciplines build and simulate increasingly big, complex and detailed models of the system they study. State-of-the-art methods no longer can be used to efficiently build, analyze and validate models because they have grown too big and too detailed. We have therefore developed new tools and indexes by carefully analyzing strengths and weaknesses of the state of the art. Where it is efficient and scalable, we use elements of known approaches (e.g., data-oriented partitioning in GIPSY, the R-Tree to find a random start element in FLAT etc.) and combine them with novel ideas (e.g., using query content in SCOUT). The resulting methods execute queries/joins faster and scale better to bigger and denser spatial models.

We have tested and deployed the indexes and techniques in the context of the Blue Brain Project (BBP [9]) where neuroscientists attempt to build and simulate the human brain. The impact of the methods we have

developed on the BBP are substantial as today they can build and analyze bigger models faster. While previously limited to building and analyzing models of 100'000 neurons, the new methods have enabled them to grow the models to 10's of millions (with a theoretical maximum of 33 million neurons on the current infrastructure). The new tools have not only accelerated discovery, but have also enabled it: analyses, e.g., assessing if the synapse density is bio-realistic, have not been possible without efficient means to access the spatial models.

Our work also demonstrates that despite decades of research in spatial data management, many challenges remain. Increasing main memory as well as novel storage technology (in the memory hierarchy), for example, means that several spatial indexes need to be redesigned. New types of datasets (e.g., dense, complex spatial datasets) make new indexes necessary and new types of queries (e.g., series of range queries) also call for the development of new indexes. Many interesting opportunities for exciting research therefore remain.

References

[1] A. Chan, R. W. H. Lau, and A. Si. A motion prediction method for mouse-based navigation. In *International Conference on Computer Graphics*, pages 139–146, 2003.

[2] V. Gaede and O. Guenther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2), 1998.

[3] S. Gnanakaran, H. Nymeyer, J. Portman, K. Y. Sanbonmatsu, and A. E. Garcia. Peptide folding simulations. *Current Opinion in Structural Biology*, 13(2):168–174, 2003.

[4] J. Gray, A. Szalay, A. Thakar, P. Kunszt, C. Stoughton, D. Slutz, and J. Vandenberg. Data Mining the SDSS SkyServer Database. In *Technical Report, MSR-TR-2002-01, Microsoft Research*, 2002.

[5] E. H. Jacox and H. Samet. Spatial join techniques. *ACM TODS*, 32(1):7, 2007.

[6] D. Komatitsch, S. Tsuboi, C. Ji, and J. Tromp. A 14.6 Billion Degrees of Freedom, 5 Teraflops, 2.5 Terabyte Earthquake Simulation on the Earth Simulator. In *Supercomputing*, 2003.

[7] J. Kozloski, K. Sfyrakis, S. Hill, F. Schürmann, and H. Markram. Identifying, Tabulating, and Analyzing Contacts Between Branched Neuron Morphologies. *IBM Journal of Research & Development*, 2008.

[8] D. Lee, J. Kim, S. Kim, K. Kim, K. Yoo-Sung, and J. Park. Adaptation of a Neighbor Selection Markov Chain for Prefetching Tiled Web GIS Data. In *Advances in Information Systems*, 2002.

[9] H. Markram. The Blue Brain Project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[10] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 2003.

[11] S. Nobari, F. Tauheed, T. Heinis, P. Karras, S. Bressan, and A. Ailamaki. TOUCH: In-Memory Spatial Join by Hierarchical Data-Oriented Partitioning. In *SIGMOD*, 2013.

[12] M.-A. Olma, F. Tauheed, T. Heinis, and A. Ailamaki. BLOCK: Efficient Execution of Spatial Range Queries in Main-Memory. Technical report, EPFL, 2013. `https://infoscience.epfl.ch/record/190731`.

[13] D.-J. Park and H.-J. Kim. Prefetch Policies for Large Objects in a Web-enabled GIS Application. *Data & Knowledge Engineering*, 37(1):65–84, 2001.

[14] M. Pavlovic, F. Tauheed, T. Heinis, and A. Ailamaki. GIPSY: Joining Spatial Datasets with Contrasting Density. In *SSDBM*, 2013.

[15] F. Tauheed, L. Biveinis, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. Accelerating range queries for brain simulations. In *ICDE*, 2012.

[16] F. Tauheed, T. Heinis, and A. Ailamaki. OCTOPUS: Efficient Query Execution on Dynamic Mesh Dataset. In *ICDE*, 2014.

[17] F. Tauheed, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. SCOUT: Prefetching for Latent Structure Following Queries. In *VLDB*, 2012.