# cTOUCH and dTOUCH: A Very Workload Balanced In-Memory Spatial Joins by Iterative Hierarchical Data-Oriented Partitioning

Sadegh Nobari Panagiotis Karras Alvis Logins
Skoltech institute of science and technology

## ABSTRACT

We are devising three novel algorithms, called reTOUCH, that are inspired by the TOUCH algorithm. Since we are dealing with big data, we need to exploit the ubiquitous parallel processors, e.g. GPU and CPU. These architectures are well designed for independent balanced workloads. Thanks to the TOUCH algorithm, the workloads are independent. However, we need balanced workloads as well. The reTOUCH algorithms are designed with the idea of creating balanced workload by reducing the asymmetry of the TOUCH algorithms as much as possible. These algorithms create indexes that produce even workloads for the actual job. In this paper we empirically examine each design and we find superiority domain of each algorithm.

## 1. PROBLEM STATEMENT

TOUCH does the spatial join of objects. Given objects of two classes we are looking for all intersections between objects of opposite classes. For example, we have some distribution of dendrites in a space. Every dendrite has its own shape and position. All dendrites are objects of first type. Then, we have objects of second type. In our example they are axons. They also have some spatial distribution. Our goal is to find such dendrites that are close enough to axons. If so, we can conclude that they are connected. Fast joining algorithm allows to reconstruct the whole net of neurones. Problem of finding close enough objects can be transformed to a problem of finding intersections by adding a sphere of radius $\epsilon$ to each object, where $\epsilon$ is the distance we are checking. The main goal of current research is to change original TOUCH algorithm in the way to allow efficient parallelization of tasks. It should be flexible to the various densities of both types. For example, if there are some place in a space where lots of objects of first dataset are located and a very few of second, then the original algorithm would be very sensitive to the order of density of these datasets. This is also connected with impossibility of parallelization of original TOUCH. Firstly, the search tree (R-tree) is build using one of types of objects and objects of other type are assigned to the nodes of this tree on the such level to minimize numbers of intersection checks with objects in the tree below the node they are assigned. Our new algorithm should remove this property and either build two separate R-trees or build one complex R-tree using both types together. This should lead to equal distribution of tasks between nodes in the search tree and possible parallelization of join phase, where we check assigned objects for intersection with underlying objects in the tree.

## 2. DESCRIPTION OF DATASETS

Data contains spatial objects from two datasets. The main goal of the algorithm is to find intersections between all objects of the first dataset with all objects from the second dataset. Lets call the affiliation of objects to datasets as types of these objects. Each object has its spatial shape, coordinates and MBR (minimum bounding box). We work with 3D data, however it could be generalized to n-dimensions. Objects are not evenly distributed in space and their shape can be any. Data about human brain will be used for experiments, which are generated by BlueGene model. The size of the objects can also differ significantly. Our algorithm must have improvements on the data with high difference between two datasets and high independent distribution of shape and position, in comparison with previous approach, where the distribution must be correlated between two datasets.

## 3. ALGORITHM

The algorithm takes spatial objects of two types and outputs the list of objects that intersects with at objects of opposite type. Algorithm contains building, assignment and joining step. Firstly, the R-tree for datasets is built, then objects are assigned to the nodes of the tree. During the last step objects assigned to nodes are checked for intersection with their children.

## 4. RESTRICTIONS

We will use only 3D and two types of objects. No restriction on spatial parameters of objects.

## 5. QUALITY

The quality criteria is the performance on the data with various types of distribution of parameters like density, object sizes etc. The faster the algorithm is (comparing with previous approaches) the better.

## 6. OPTIMIZATION

There are number of parameters, which must be tuned for effective work. These are: maximum height of possible assignment, fanout (number of objects in a node), maximum capacity of a node on assignment step, number of objects per bucket.

# 7. MODEL FOR EXPLOITATION

The final product will be the command-line program with two datasets for input (list of coordinates and type). Output will be the performance of the algorithm and the list of intersections.

# 8. BASIC ALGORITHM

## 8.1 Original TOUCH algorithm

The original TOUCH algorithm consists of building R-Tree step, Assignment step and Joining step.

**Algorithm 1** TOUCH algorithm, building R-Tree step

```
1: function CREATEPARTITIONS(dataset_A)
2:     nextInput ← [dataset, level = 0]
3:     while dataset is not empty do
4:         [dataset, level] ← nextInput
5:         Sort(dataset)
6:         split dataset into parts ds with size of nodeSize
7:         for all ds in dataset do
8:             totalMBR = new MBR
9:             for all object in ds do
10:                totalMBR ← objectMBR          ▷ expand MBR of current node
11:            tree ← [object, level]
12:            nextInput ← new object(totalMBR, level = size of tree)
```

**Algorithm 2** TOUCH algorithm, Assignment step

```
1: function ASSIGNMENT(dataset_B)
2:     for all b in dataset_B do
3:         currentNode =top(tree)
4:         loop
5:             for all object in currentNode do
6:                 if b intersects one object then
7:                     currentNode = object
8:                 if b interects more than one object then
9:                     assign b to currentNode
```

**Algorithm 3** TOUCH algorithm, Join step

```
1: function PROBE(tree)
2:     for all node in tree do
3:         if number of objects from dataset_B in node > 0 then
4:             for all child leafs of node do
5:                 join(leaf, node)          ▷ Plane-sweeping join algorithm
```

## 8.2 dTOUCH (double TOUCH)

This modification is based on building two separate R-trees. First step is building R-tree using objects from dataset A. dTOUCH has the building step the same as original TOUCH algorithm (ref. 1). After it, we assign objects of dataset B to this tree. During the assignment phase the level of the node where we decide to assign our object represents the workload for joining step. Lower level corresponds to easier joining because every assigned object must be checked with all leafs - descenders and number of descenders grows exponentially with the level. Here the key idea for dTOUCH goes: lets restrict assignment to high levels of the tree. Now we do the assignment of objects of dataset B to the tree built from objects of dataset A the same way as in original TOUCH but only to the levels below some parameter MAX_LEVEL.

**Algorithm 4** dTOUCH algorithm, Assignment restriction part

```
1: function ASSIGNMENT(dataset_B)
2:     for all b in dataset_B do
3:         currentNode =top(tree)
4:         loop
5:             for all object in currentNode do
6:                 if b intersects one object then
7:                     currentNode = object
8:                 if b intersects more than one object then
9:                     if level of object <= max_level then
10:                        assign b to currentNode
11:                        delete b from dataset_B
```

After we have done assignment step, we have some objects B filtered, some assigned and some ignored due to level restriction. Ignored data can probably intersect with some of objects of opposite type. This ignoring corresponds to the situation when the size of object B in space is much larger than all objects of dataset A or just intersect with considerable amount of them. In this case it is more efficient to change roles of A and B dataset and assign objects of dataset A. Sequentially, lets build one more R-tree using this ignored data from dataset B and assign all objects from dataset A to this new small tree. This time do not use any restrictions on level avoiding any result loss. Finally, do the joining using both trees the same way as in original TOUCH.

## 8.3 cTOUCH (complex TOUCH)

The key idea of rewriting TOUCH code is making the algorithm symmetric to A and B datasets. In this implementation we will try to build one symmetric R-tree, using both datasets as equally as possible. For pretty illustrating the principals of the approach lets introduce some colors and definitions. Let objects ($o$) from dataset A be red objects ($o$) and from dataset B - blue objects ($o$). Each objects has its own MBR ($\square$) that has corresponding color. Let MBR of the node $i$ on the level $l$ be red ($v_i^l \leftarrow \square_i^l$) if it was built as intersection of all red MBRs of the children ($\square_i^l = \bigcup \square_{i-1}^l$). Lets MBR of the node be blue if it was built from blue ones. Let MBR be black ($\square_i^l$) if it was build as intersection of blue and red MBRs. Let the assigned red objects be dark red ($\blacksquare_i^l$) and assigned blue - dark blue ($\blacksquare_i^l$).

### 8.3.1 Overview

Finished with colors, now the introduction to the algorithm goes. The algorithm consists of the same steps as original TOUCH: building, assignment and joining steps. During the building step we take all objects of both colors, build leaf nodes (create initial groups of objects) and build one mutual R-tree using Hilbert curve for indexing starting

from leafs and creating two MBR per each node. Assignment phase is about taking all the objects from the leafs of the tree and assigning them to the tree traversing from the root to the leafs. At the joining step for each node we check the intersection of assigned object with all assigned objects which were assigned to the descenders of the node.

*Reminder.*
$\square_i^l$ - MBR where $l$ - level in the tree, $i$ - index of the node where it must be located.

### 8.3.2 *Building phase*

See the algorithm 5.

**Algorithm 5** cTOUCH algorithm, Building phase

$\square_i^0 \leftarrow o_i$
$\square_i^0 \leftarrow o_i$
$\forall j \quad v_j^0 \leftarrow \square_i^0, \square_i^0 \quad i \in \{jC, jC+1, ..., (j+1)C\}$ ▷ $C$ - leaf capacity
$\forall v_j^0 \quad \square_j^0 \leftarrow \square_j^0 \cup \square_j^0$
$\Omega \leftarrow v_j^0 \quad \forall j$
**while** size$(\Omega) \neq 1$ **do**
  $v_i^l \leftarrow \Omega$
  Sort $v_i^l$ by $\square_i^l$ using Hilbert curve
  $\forall n \quad \omega_n \leftarrow v_i \quad i \in \{jF, j(F+1), ..., (j+1)F\}$ ▷ $F$ - fanout
  **for all** $j$ **do**
    **for all** $\square_i^l, \square_i^l \leftarrow \omega_n$ **do**
      $\square_j^{(l+1)} \leftarrow \bigcup_i \square_i^l$
      $\square_j^{(l+1)} \leftarrow \bigcup_i \square_i^l$
      $v_j^{(l+1)} \leftarrow \square_j^{(l+1)}, \square_j^{(l+1)}$
      $E \leftarrow (i, j)$ ▷ $E$ - set of edges of the tree
      $\Omega \leftarrow v_j^{(l+1)}$
Root $\leftarrow \Omega$

### 8.3.3 *Assignment phase*

See the algorithm 6.

### 8.3.4 *Joining phase*

See the algorithm 7.

## 8.4 Cost of object

In both cTOUCH and dTOUCH implementation each object has a cost $Z$. This number shows approximate upper bound for number of comparisons of this object with every object of opposite type in the descender nodes during the Join phase. Due to independence of the processing of each object during the joining phase, they can be distributed among processors and must be distributed equally in the meaning of workload. Having such cost number we can implement greedy algorithm to distribute tasks between processors. Cost is calculated using the information about the number of MBRs which intersects with the object. This number is calculated during the assignment phase (for all implementations of TOUCH) and it must be calculated anyway either we use cost estimation or not. So, we do not increase time for assignment step. Given the number of intersections with MBRs of children, $c$, lets estimate cost for cTOUCH and dTOUCH.

*dTOUCH.*

**Algorithm 6** cTOUCH algorithm, Assignment phase

**for all** $v_i^0$ **do**
  **for all** $o \leftarrow v_i^0$ **do**
    $v_j^l$ = Root
    **loop**
      $\omega = \{v_k^{l-1} : o^r \cap (\square_k^{(l-1)} \cup \blacksquare_k^{(l-1)}) \neq \varnothing, (k, j) \in E\}$
      **if** $\omega = \varnothing$ **then**
        Filter $o$
        Break
      **if** size$(\omega) = 1$ **then**
        $v_j^l \leftarrow \omega$
      **if** size$(\omega) > 1$ or $v_j^l$ - leafnode **then**
        $v_j^l \leftarrow o$
        **for** $l' \geq l$ **do**
          $\blacksquare_k^{l'} \leftarrow \square_j^l \quad \forall (k, j) \in E$
        Break
  Delete $\square_i^0$
  **for all** $l > 0$ and $k : (k, i) \in E$ **do**
    $\square_k^l \leftarrow \bigcup_j \square_j^{(l-1)} \quad \forall j : (j, k) \in E$
  **for all** $o \leftarrow v_i^0$ **do**
    $v_j^l$ = Root
    **loop**
      $\omega = \{v_k^{l-1} : o \cap (\square_k^{(l-1)} \cup \blacksquare_k^{(l-1)}) \neq \varnothing, (k, j) \in E\}$
      **if** $\omega = \varnothing$ **then**
        Filter $o$
        Break
      **if** size$(\omega) = 1$ **then**
        $v_j^l \leftarrow \omega$
      **if** size$(\omega) > 1$ or $v_j^l$ - leafnode **then**
        $v_j^l \leftarrow o$
        **for** $l' \geq l$ **do**
          $\blacksquare_k^{l'} \leftarrow \square_j^l \quad \forall (k, j) \in E$
        Break
  **for all** $l > 0$ and $k : (k, i) \in E$ **do**
    $\square_k^l \leftarrow \bigcup_j \square_j^{(l-1)} \quad \forall j : (j, k) \in E$

**Algorithm 7** cTOUCH algorithm, Joining phase

**for all** $v_i^l$ **do**
  **for all** $o \leftarrow v_i^l$ **do**
    **for all** $v_j^{l'} : l' \leq l, j \in E$ **do**
      **for all** $o \leftarrow v_j^{l'}$ **do**
        **if** $o \cap o \neq \varnothing$ **then**
          Result $\leftarrow (o, o)$
  **for all** $o \leftarrow v_i^l$ **do**
    **for all** $v_j^{l'} : l' \leq l, j \in E$ **do**
      **for all** $o \leftarrow v_j^{l'}$ **do**
        **if** $o \cap o \neq \varnothing$ **then**
          Result $\leftarrow (o, o)$
  **for all** $o, o \leftarrow v_i^l$ **do**
    **if** $o \cap o \neq \varnothing$ **then**
      Result $\leftarrow (o, o)$

Here objects are assigned in nodes but can be joined only with objects in leafs. So, given the level of our assigned object $l$, we have $cF^l$ leafs each containing $C$ objects. Here, we assume that on each level below our object intersect with every MBR of opposite type and there are $F$ (fanout) MBRs on each level.

$$Z = cCF^l$$

### *cTOUCH.*

Here objects can be joined at every node and every level of the tree. So, we will calculate the number of possible comparisons during the assignment step. Each time we assign an object to the node, we add 1 to every assigned object of the opposite type in every ancestor of the node.

$$Z(o_i \leftarrow v_i^l) = N\{o_j^b \leftarrow v_j^{l'} : (i,j) \in E \quad \& \quad l' > l\}$$

## 9. REFERENCES


[1] W. G. Aref and H. Samet. A Cost Model for Query Optimization Using R-Trees. In *GIS '94*.

[2] W. G. Aref and H. Samet. Cascaded Spatial Join Algorithms with Spatially Sorted Output. In *GIS '96*.

[3] W. G. Aref and H. Samet. Hashing by Proximity to Process Duplicates in Spatial Databases. In *CIKM '94*.

[4] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The Priority R-tree: a practically efficient and worst-case optimal R-tree. In *SIGMOD '04*.

[5] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. Scalable Sweeping-Based Spatial Join. In *VLDB '98*.

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: an efficient and robust access method for points and rectangles. *SIGMOD Record*, 19(2):322–331, 1990.

[7] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-Trees. In *SIGMOD '93*.

[8] J.-P. Dittrich and B. Seeger. Data Redundancy and Duplicate Detection in Spatial Join Processing. In *ICDE 2000*.

[9] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 3rd edition, 2000.

[10] A. Farris, A. Sharma, C. Niedermayr, D. Brat, D. Foran, F. Wang, J. Saltz, J. Kong, L. Cooper, T. Oh, T. Kurc, T. Pan, and W. Chen. A Data Model and Database for High-resolution Pathology Analytical Image Informatics. *Journal of Pathology Informatics*, 2(1):32, 2011.

[11] Y. J. García, M. A. López, and S. T. Leutenegger. A Greedy Algorithm for Bulk Loading R-trees. In *GIS '96*.

[12] 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.

[13] A. Guttman. R-trees: a Dynamic Index Structure for Spatial Searching. In *SIGMOD '84*.

[14] O. P. Hamill, A. Marty, E. Neher, B. Sakmann, and F. J. Sigworth. Improved Patch-clamp Techniques for High-resolution Current Recording from Cells and Cell-free Membrane Patches. *Pflügers Archiv European Journal of Physiology*, 391:85–100, 1981.

[15] E. H. Jacox and H. Samet. Spatial Join Techniques. *ACM TODS '07*.

[16] I. Kamel and C. Faloutsos. Hilbert R-tree: An Improved R-tree using Fractals. In *VLDB '94*.

[17] N. Koudas and K. C. Sevcik. Size Separation Spatial Join. In *SIGMOD '97*.

[18] J. Kozloski, K. Sfyrakis, S. Hill, F. Schürmann, C. Peck, and H. Markram. Identifying, Tabulating, and Analyzing Contacts Between Branched Neuron Morphologies. *IBM Journal of Research and Development*, 52(1/2):43–55, 2008.

[19] S. Leutenegger, M. Lopez, and J. Edgington. STR: a Simple and Efficient Algorithm for R-Tree Packing. In *ICDE '97*.

[20] M.-L. Lo and C. V. Ravishankar. Spatial Hash-Joins. In *SIGMOD '96*.

[21] M.-L. Lo and C. V. Ravishankar. Spatial Joins Using Seeded Trees. In *SIGMOD '94*.

[22] G. Luo, J. F. Naughton, and C. J. Ellmann. A non-blocking parallel spatial join algorithm. In *ICDE*, pages 697–705, 2002.

[23] N. Mamoulis and D. Papadias. Slot Index Spatial Join. *IEEE TKDE*, 15(1):211–231, 2003.

[24] P. Mishra and M. H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, 24(1):63–113, 1992.

[25] S. Nobari, T.-T. Cao, P. Karras, and S. Bressan. Scalable parallel minimum spanning forest computation. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 205–214, New York, NY, USA, 2012. ACM.

[26] J. Orenstein. A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces. In *SIGMOD '90*.

[27] J. M. Patel and D. J. DeWitt. Partition Based Spatial-Merge Join. In *SIGMOD '96*.

[28] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer, 1993.

[29] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *VLDB '87*.

[30] M. Ubell. The Montage Extensible DataBlade Architecture. In *SIGMOD '94*.