Configuring Spatial Grids for Efficient Main Memory Joins

Farhan Tauheed, Thomas Heinis, and Anastasia Ailamaki

École Polytechnique Fédérale de Lausanne (EPFL), Imperial College London

Abstract. The performance of spatial joins is becoming increasingly important in many applications, particularly in the scientific domain. Several approaches have been proposed for joining spatial datasets on disk and few in main memory. Recent results show that in main memory, grids are more efficient than the traditional tree based methods primarily developed for disk. The question how to configure the grid, however, has so far not been discussed.
In this paper we study how to configure a spatial grid for joining spatial data in main memory. We discuss the trade-offs involved, develop an analytical model predicting the performance of a configuration and finally validate the model with experiments.

1 Introduction

Spatial joins are an operation of increasing importance in many applications. Whether for spatial datasets from astronomy, neuroscience, medicine or others, the join has to be performed to find objects that intersect with each other or are within a given distance of each other (distance join). An efficient execution of this operation is therefore key to improve overall performance.

In this context main memory joins are becoming increasingly important because many datasets fit into the main memory directly. Even if they do not, and the join has to be performed on disk, a crucial part of a disk-based join is the in memory join. While the strategies of disk-based approaches to partition the data (replication or no replication, space-oriented partitioning or data-oriented partitioning) so it fits into memory differ [1], every approach requires an efficient algorithm to join two partitions in main memory.

The only approaches specifically designed to join spatial data in memory are the nested loop join and plane sweep join approach. The nested loop join technique works by comparing all spatial elements pairwise and is thus computationally very expensive. The plane sweep approach [2] sorts the datasets in one dimension and scans both datasets synchronously with a sweep plane. It has has a lower time complexity but compares objects no matter how far apart they are on the sweep plane.

To speed up the join time over these two slow approaches, tried and tested tree-based indexing techniques on disk have been optimized for main memory. Although these approaches indeed improve performance, recent research shows that a simple grid performs best to join for one-off spatial joins in memory [3]. The problem of configuring the grid optimally, however, is challenging and remains unaddressed to date.

In this paper we therefore develop a cost model that can be used to configure the grid optimally. With experiments we show that the cost model can accurately predict the performance of the join.

2 Grid-based Spatial Join

Spatial joins are typically split into two phases, filtering and refinement [4]. The filtering phase uses a coarse grained collision detection and finds intersections between approximations of the actual objects. The refinement phase, is used to remove the false positive by using an exact, but time-consuming object-object collision test. The refinement phase is a computationally costly operation with little room for improvement and so, like other approaches for spatial joins [4], the spatial grid-based spatial join focuses on improving the filtering phase.

On a high level, the grid-based spatial join tackles the filtering phase with a three dimensional uniform grid and uses it as an approximate method to group spatially close objects. In the building step we map the MBRs of the objects of both datasets on a grid while the probing step retrieves MBRs from the same cells (which are thus close together) and compares them pairwise.

In the following we first explain the two steps, i.e., building and probing.

2.1 Building Step

The algorithm iterates over both sets of objects and for each object calculates its MBR and maps it on the uniform grid. The mapping process finds the grid cells that intersect with the MBR volume and creates a pointer from each intersecting grid cell to the MBR. By using a uniform grid we simplify the calculation of the mapping of MBR to cell and can calculate the list of intersecting grid cells efficiently as follows. First, we use the minimum and maximum coordinate of the MBR to find the minimum and maximum grid cell intersecting it. Second, we use a nested loop to iterate over all the grid cells bounded by these minimum and maximum grid cell.

The mapping of MBR to grid cells is ambivalent as one MBR can also map to several grid cells. To store this many-many relationship between the MBR and the grid cell we use a hashmap which maps a list of pointers to a grid cell identifier. To map an MBR we access the list of pointers for each intersecting grid cell and insert the pointer in their respective list. Apart from providing a fast access mechanism to the list of pointers the hash table also helps to reduce the memory footprint as only grid cells containing one or more pointers to an MBR require an entry in the hash map and no memory is wasted in storing empty grid cells.

2.2 Probing Step

The probing step of the algorithm retrieves the mapped MBRs of the two objects sets from the grid. The algorithm iterates over all the grid cells and separately retrieves all the MBRs in the cell. For each cell it then compares all MBRs representing objects from the first dataset with all MBRs representing objects from the second dataset.

Because MBRs can be mapped to several cells, intersections between the same pair of MBR's may be detected multiple times. This leads to a) additional computational overhead (because of additional comparisons) and b) duplicate results. The spatial grid-based spatial join thus use a global (across all grid cells) set based data structure in a postprocessing step to deduplicate the results before reporting them.

3 Configuring the Grid-based Spatial Join

As we discuss in the following, the performance of the algorithm we propose for filtering depends on the configuration, i.e., the grid resolution used, as well as the data distribution.

3.1 Impact of Data Skew

Uniform grids are very sensitive to data skew and using them in spatial join algorithm can lead to performance degradation because in dense regions the number of MBRs mapped on a grid cell increases and consequently the number of comparisons required increases too. All MBRs may be mapped to one single grid cell. In this scenario the performance of spatial grid hash join becomes equivalent to a nested loop join because all MBRs need to be compared pairwise and the total number of comparisons is $O(n^2)$. Even worse, all MBRs may be mapped to the same multiple cells and the nested loop is executed comparing the same MBRs several times (resulting in duplicates that need to be eliminated).

The problem of data skew can be addressed by setting a finer grid resolution. With a finer grid resolution also the objects or their MBRs in very dense regions of the datasets will be distributed to numerous grid cells instead of just a few. As we will discuss in the next section, the resolution, however, cannot be set infinitely fine-grained, but reducing the cell size still helps to address the problem of data skew.

3.2 Impact of Grid Resolution

Changing the grid resolution, i.e., making it coarser or finer grained directly affects the performance of the algorithm.

In case of a fine resolution, an MBR is likely to be mapped to many grid cells and the memory footprint therefore increases. This also leads to degraded performance because more comparisons need to be performed in the probing phase. The number of comparisons increases because MBRs mapped to several cells need to be compared more than once.

In case the resolution is coarse, each grid cell contains many MBRs and hence the performance degrades because all MBRs in the same grid cell need to be compared pairwise, thereby increasing the number of comparisons. A coarse resolution, however, lowers the memory consumption of the algorithm because an MBR is less likely to be mapped to many grid cells, thereby reducing duplication (even if pointers are used). Additionally, the probability of comparing the same pair of MBRs several times because they are assigned to several cells (as is the

problem of a fine resolution) is considerably reduced, reducing the overall number of comparisons.

Both extremes have advantages and disadvantages and it is difficult to set the resolution intuitively. In the following we therefore develop an analytical model that will predict the optimal grid resolution for two sets of objects in terms of number of total comparisons.

3.3 Analytical Model

To determine the optimal resolution we develop a cost model for predicting the time for the join. Like our algorithm we also split the cost model into building and probing costs.

Building Cost The building phase loops over the MBR of each of the N_d objects in the first dataset and for each MBR finds the intersecting grid cells using the *getCell* (gC) function. For each cell a *hashLookup* (hL) is performed to obtain the list of pointers that point to the MBR and in the end the pointer of the current MBR is added to the list using *insertPointer* (iP). The resulting cost is summarized in the following equation with C_i as the number of cells an MBR intersects with:

$$BuildingCost = \sum_{i=1}^{N_d} \left\{ gC(MBR_i) + \sum_{j=1}^{C_i} [hL(j) + iP(\&MBR_i)] \right\} \tag{1}$$

The cost of the *getCell* (gC) function is defined as follows:

$$gC(MBR_i) = \sum_{j=1}^{C_i} vertexToGridCell(j)$$

To determine the actual building cost we need to know the duration of each individual operation and the number of iterations of each loop. *vertexToGridCell* and *insertPointer* both are constant time operations and for the sake of simplicity we also assume *hashLookup* to be a constant time operation (this essentially means we use a tuned hash table which is collision-free). The execution time of all these operations heavily depends on the hardware platform they are executed on. We use microbenchmarks to determine their execution time.

N_d, the number of objects in the first dataset, is a given and *Average(C)*, the number of cells an average object's MBR maps to, is calculated as follows (instead of calculating C_i for each object we use an approximation, i.e., *Average(C)*).

Average(C), the number of cells an average MBR_i maps to depends on the average volume of the MBR and on the grid resolution. On average, the MBR of an object particular MBR_i the number of cells it is mapped to, can be approximated by $Volume(MBR_i)/Volume(gridCell)$. This, however, is only an approximation and it underestimates the number of grid cells because the exact number of grid cells intersecting depends on the exact loation of MBR_i relative

to the grid cells. If MBR_i is exactly aligned with the grid cell then the combined volume of the grid cell is equal to the volume of MBR_i. If, however, MBR_i is not aligned, then the combined volume of the grid cell is greater than the volume of MBR_i to at most the volume of a single grid cell.

To resolve this issue we expand the volume of MBR_i by half the volume of a single grid cell, to get a better approximation for the average case.

$$Total(C) = \sum_{i=1}^{N_d} \left\{ \frac{Volume(MBR_i) + Volume(gridCell)/2}{Volume(gridCell)} \right\}$$

$$Average(C) = Total(C)/N_d$$

Probing Cost Similar to the building step, the probing step loops over each object in the second dataset. For each object the algorithm finds the list of cells intersecting the MBR of the object. However, instead of mapping the MBR on the grid, the probing step retrieves the mapped MBRs from the first dataset for testing the overlap.

$$ProbingCost = \sum_{i=1}^{N_a} \left\{ gC(MBR_i) + \sum_{j=1}^{C_i} \left[hL(j) + \sum_{k=1}^{S_j} (oT(i,k) + dD()) \right] \right\} \quad (2)$$

The operations of the probing step are *overlapTest* (oT), which compares two MBRs for overlap, and *deduplication* (dD), which uses a hash based set to remove duplicate results. We consider both these operations as constant time operations, because we assume a near collision free hash set for our estimates. The number of iterations of the loop N_a is the size of the outer data set.

Similar to the building cost model, we use *Average(C)* to approximate the number of grid cells that intersect with the MBRs of the outer data set.

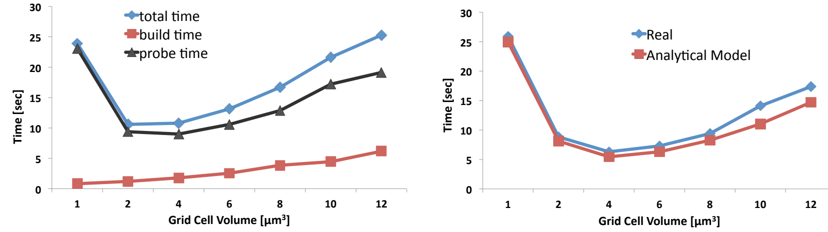
To estimate S_j we use an approximation *Average(S)*. *Average(S)* is the number of first dataset objects mapped to grid cells, but only the grid cell which intersects the MBRs of objects of the second dataset.

The probing step typically takes the majority of the the total time of the join. Setting the resolution optimally therefore has a substantial impact on the performance of the overall algorithm. By using a increasingly fine resolutions, the cell volume decreases, this increases the number of grid cells that intersect the MBR of the outer dataset and hence the performance degrades. At the same time, however, the number of overlap test comparisons decreases because we do not compare objects for overlap which are not located spatially close.

3.4 Optimal Grid Resolution

The sum of both cost models is a concave up curve and the local minimum and hence the optimal value is where the first derivative is equal to zero. To validate the model we have tested it using experiments where we vary the grid cell size. For the experiments we use neuroscience data where 4.5 million cylinders model 1692 neurons and we use the experimental setup described in [5].

In Figure 1 (a) we measure the individual components (build & probe) as well as the total time of the join. Clearly, for both components there is an optimal (at the same grid cell size) where the join is executed the fastest. The second experiment (Figure 1 (b)) plots the total execution time against the analytical model and shows that the the model can indeed be used to accurately predict the performance and thus to determine the grid configuration.



(a) Time for grid-based spatial join operations.

(b) Analytical model compared to measurements.

Fig. 1. Validating the analytical model of the grid-based spatial join.

4 Conclusions

Whether in disk- or in memory spatial joins, the main memory join is a crucial operation. Recent research demonstrated that grid-based approaches outperform tree-based ones in main memory [3], but the question of how to set the optimal resolution remains unaddressed. In this paper we described our implementation of a grid-based spatial join and, crucially, developed and analytical model to predict performance. Our experimental results show that with little information about the datasets to be joined as well as the underlying hardware, the model accurately predicts performance. While it may be difficult to estimate the execution of individual operations, microbenchmarks can be used to find accurate values. Even in the absence of the cost of the operations, the model can still give insight into how to configure the grid for optimal performance.

References

1. Jacox, E.H., Samet, H.: Spatial Join Techniques. ACM TODS '07
2. Preparata, F., Shamos, M.: Computational Geometry: An Introduction. Springer (1993)
3. Šidlauskas, D., Jensen, C.S.: Spatial Joins in Main Memory: Implementation Matters! In: VLDB '15
4. Orenstein, J.: A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces. In: SIGMOD '90
5. Tauheed, F., Biveinis, L., Heinis, T., Schürmann, F., Markram, H., Ailamaki, A.: Accelerating range queries for brain simulations. In: ICDE '12