Parallel Selectivity Estimation for Optimizing Multidimensional Spatial Join Processing on GPUs

Jianting Zhang
Dept. of Computer Science
The City College of New York
New York, NY, USA
jzhang@cs.ccny.cuny.edu

Simin You
Dept. of Computer Science
CUNY Graduate Center
New York, NY, USA
syou@gradcenter.cuny.edu

Le Gruenwald
Dept. of Computer Science
The University of Oklahoma
Norman, OK, USA
ggruenwald@ou.edu

***Abstract*—Processing large-scale data is typically memory intensive. The current generation of Graphics Processing Units (GPUs) has much lower memory capacity than CPUs which is often a limiting factor in processing large-scale data on GPUs. It is desirable to reduce memory footprint in spatially joining large-scale datasets through query optimization. In this study, we present a parallel selectivity estimation technique for optimizing spatial join processing on GPUs. By integrating the multi-dimensional cumulative histogram structure and the summed-area-table algorithm, our data parallel selectivity estimation technique can be efficiently realized on GPUs. Experiments on spatially joining two sets of Minimum Bounding Boxes (MBBs) derived from real point and polygon data, each with about one million MBBs, have shown that selectivity estimation at four grid levels took less than 1/3 of a second on a Nvidia GTX Titan GPU device. By using the best grid resolution, our technique saves 38.4% memory footprint for the spatial join.**

Keywords— Selectivity Estimation, Spatial Join, GPU, Parallel Design, Cumulative Histogram

I. INTRODUCTION

Spatial data volumes are fast increasing due to advances of locating, sensing and simulation techniques. For example, although navigation devices (e.g. GPS, cellular and WIFI network-based ones) embedded in smartphones (nearly 1.5 billion sold in 2015 [1]) have already generated large volumes of location and trajectory data, the next generation of consumer electronics, such as Google Glasses and Microsoft HoloLens, are likely to generate even larger volumes of location-dependent multimedia data. Objects identified from high-resolution satellite imagery and medical imagery, when represented as vectors of geometric coordinates, can also be considered as spatial data. In addition, large-scale climate, astronomical and molecular simulations are likely to produce even larger spatial datasets. Very often different spatial datasets need to be joined to derive new information and knowledge to support decision making. For example, GPS traces can be better interpreted when aligned with urban infrastructures, such as road networks and Point of Interests (POIs), through spatial joins. As spatial datasets are getting increasingly larger, techniques for high-performance spatial join processing on commodity and inexpensive parallel hardware become crucial in addressing the "BigData" challenge.

Spatial joins can be considered as extensions of relational theta joins [2] where spatial relationships, such as distance and topology, are involved in the joining criteria [3]. While considerable research on join query processing for both relational [2] and spatial [3] data has been reported, including that targeted for parallel and distributed computing platforms [3], there is little research on parallel spatial joins on GPUs that are capable for general computing. Compared with multi-core CPUs, the current generations of GPUs typically have limited memory capacity (generally in the order of a few GBs), which frequently becomes a constraining factor for parallel spatial joins on large-scale spatial datasets. In addition, different from multi-core CPUs that are designed to support coarse-grained task-level parallelisms, fine-grained data parallelisms are crucial in achieving hardware potentials on GPUs. As such, many existing spatial join techniques that are either sequential in nature or rely on coarse-grained parallelisms can be inefficient when applied to GPUs. The combined technical challenges in minimizing memory footprints and maximizing data parallelisms have motivated us to develop novel spatial join techniques on GPUs.

In our previous studies, we have explored several GPU-based techniques for parallel spatial join processing, such as distance based point-to-polyline join [4], trajectory similarity join [5], and topology based point-in-polygon test based spatial join [6]. These GPU-based techniques adopt the classical two-phase spatial join framework, i.e., a filtering phase followed by a refinement phase [3]. While the refinement phase typically involves more floating point computation and is desirable to utilize GPUs for speeding up, it is more technically challenging in improving the efficiency of the filtering phase on GPUs under stricter resource constraints, e.g., GPU memory capacity. Compared with the refinement phase that can utilize batch processing to reduce resource requirements in a single batch, it is more difficult to explore a similar strategy for filtering, as global information is typically required in this phase. Spatial filtering techniques that minimize memory consumption are thus preferred from an implementation perspective.

While spatial indexing techniques, such as pre-built quad-trees and R-Trees [7], have been frequently used to speed up spatial joins in classic computing models that are designed for serial algorithms, uniprocessors and disk-resident

systems, their suitability on GPUs for spatial joins needs to be reevaluated. First of all, very often their hierarchical tree structures and irregular memory access patterns incur significant performance penalty on parallel hardware, especially GPUs. Second, the complex data structures are expensive to construct and maintain and difficult to manipulate. Utilizing multi-dimensional histograms as light-weighted data structures that are parallelization friendly to facilitate spatial join processing on GPUs is thus desirable in many cases. Different from heavy-weighted spatial index structures that are associated with data items for direct query processing, these histograms contain only essential statistical information to guide the process of choosing optimal/suitable parameters for spatial joins under resource constraints, with or without using additional spatial indices. Indeed, as discussed in Section II, selectivity estimation is an important component in efficiently processing spatial joins, especially when spatial indexes are not available.

Our parallel selectivity estimation technique is based on Cumulative Histogram (CD) to count the numbers of Minimum Bounding Boxes (MBBs) that intersect with cells in uniformly tessellated grids, compute the possible numbers of pairs in the grid cells, and, *choose an optimal grid resolution that satisfy GPU memory footprint budget*. While cumulative histograms have been utilized for spatial query processing in a previous study [8], we believe we are the first to take advantage of data parallelisms in constructing and utilizing CDs for parallel selectivity estimation to guide spatial joins on GPUs. Furthermore, our design seamlessly integrates the well-received Summed-Area-Table (SAT) algorithm in computer vision and image processing domain [9]. Based on the idea, we are able to provide a simple design and implementation that can be presented as a chain of several parallel primitives [18], i.e., sort/scan/reduce/scatter/transform, which are well understood in the parallel computing community and well-supported across multiple parallel hardware platforms, including Nvidia and AMD GPUs. We have further optimized some of the parallel primitives in the specific application context to improve overall performance.

Our technical contributions can be summarized as follows:

1) We present a novel parallel design and implementation of parallel selectivity estimation on GPUs by integrating a cumulative histogram and the Summed-Area-Table algorithm.

2) We perform preliminary experiments on real spatial datasets to demonstrate the effectiveness and efficiency of the proposed technique.

The rest of the paper is arranged as follows. Section II introduces the background, motivation and related work. Section III provides the parallel spatial join framework. Section IV presents the details of the parallel design and implementation. Section V is the experiments, and finally, Section VI is the conclusion and future work.

II. Background and Related Work

Given two spatial datasets each with a geometrical attribute *the_geom*, i.e., T1(*id*, *the_geom*) and T2(*id*, *the_geom*), the basic form of spatial join processing can be expressed as the following SQL statement:

SELECT * from T1, T2

WHERE *ST_OP* (T1.*the_geom*, T2.*the_geom*)

Here the geometric attributes in T1 and T2 can be any of the geometric types (e.g., point, polygon and polyline) and *ST_OP* can be any of the spatial relationships (e.g., intersects, within) defined by the Open Geospatial Consortium (OGC) Simple Feature Specification (SFS) [10] which has been the cornerstone of virtually all commercial (e.g., Oracle Spatial and Microsoft SQL Server Spatial) and open source (Postgresql/PostGIS) spatial databases. More complex queries may also involve additional attributes in T1 and T2, additional operators (e.g., *count*, *sum*) and additional clauses (e.g., *group by*, *having* and *order by*). Similar to theta joins in relational queries, spatial joins can be conceptually formulated as Cartesian products followed by evaluating spatial relationships between two geometrical objects based on some well-established principles (e.g., nearest neighbor) and/or computational geometry algorithms (e.g., point-in-polygon test). Assuming the cardinality of T1 and T2 to be n1 and n2, respectively, similar to processing relational joins, indices can be constructed to reduce the complexity from O(n1*n2) to O(n1) or O(n2) provided that a good spatial filtering strategy is available so that a spatial object in T1 will only be paired with a limited number of spatial objects in T2. As argued in [3], spatial joins are distinguished from relational joins due to the fact that spatial data are inherently multi-dimensional data that exhibit several unique features, e.g., lacking total ordering that preserves proximity (which makes sort-merge join largely inapplicable and/or inefficient), unsuitable for grouping due to having spatial extents (which makes equijoin inapplicable), and, requiring complex geometric computation (which is typically much more expensive than arithmetic operations for relational joins).

Hundreds of indexing structures have been developed in the past few decades to index and query spatial data [7]. In addition, we refer to the excellent survey paper [3] for a comprehensive review on spatial join techniques, including several parallel and distributed spatial join techniques on traditional cluster computing environments. We also refer to several recent works on spatial join processing on MapReduce/Hadoop clusters [11,12] with demonstrated scalability at the expense of singe node efficiency. Despite that shared-memory systems are getting increasingly popular and affordable in both personal and cluster computing settings and typically are easier to program, almost all existing parallel spatial join techniques are designed for shared-nothing architectures (see [13] for a recent brief survey). While shared-nothing architectures are generally considered to have better scalability, recent studies show that parallel data processing on MapReduce/Hadoop clusters typically is only able to utilize a fraction of hardware resources, mostly due to disk I/O and network bandwidth constraints [14]. As GPUs that are capable of general computing typically have large numbers of processors (in the order of 10^3), much higher

bandwidth (300-500 GB/s) and more floating point computing power (by design) [15][16], an alternative to cluster computing in solving moderate sized spatial join problems on single GPU devices becomes promising [4,5,6]. We note that as GPUs are typically used as accelerators in computing nodes, it is quite possible to integrate the two sets of techniques to solve larger scale spatial join problems when needed.

Selectivity estimation is considered a vital component in query optimization in both relational databases and spatial databases. Given a set of query items in T1, selectivity estimation techniques estimate the numbers of items in T2 that are likely to be joined with each of the query items. Fast and accurate selectivity estimations can help database query optimizers to choose better query plans under resource constraints. Some techniques on selectivity estimation for spatial joins and other types of spatial query processing [8, 19-41] have been reported.

Several existing techniques derive histograms by querying data or indices directly, e.g., [35][41]. These techniques enjoy the flexibility of using arbitrary bin shapes and can be optimized with certain properties [40]. However, as the MBBs of non-point geometric objects (including aggregated points) are variable, these techniques require loops over bins along both width and height in 2D spaces. It is more difficult to provide a simple parallel design and an efficient implementation for these techniques as they are not parallelization friendly. In addition, many of these techniques are almost as expensive as spatial indexing and spatial queries and they may incur significant computing overheads when the numbers of bins are large. While materialized histograms can be used to speed up spatial joins effectively, it might be too expensive to construct in an on-demand manner.

In this study, we are interested in selectivity estimation techniques that are based on regularly spaced multi-dimensional histograms (e.g., [19, 26, 27, 31, 33]) for two reasons. First, when the histogram bins use the same configuration for both datasets involved in a spatial join, the total numbers of pairs to be processed in the refinement phase in each bin can be easily calculated as $|B1_i|*|B2_i|$ where $|B1_i|$ and $|B2_i|$ are the numbers of objects that intersect with spatial extent of the common bin (i.e., grid cell) in T1 and T2, respectively. Second, regularly spaced multi-dimensional histograms are more parallelization friendly and are likely to be more efficient on modern parallel hardware to a certain extent (even through hierarchical tree structures might be more desirable at upper levels for efficiency concerns).

By observing that generating a 1D cumulative histogram is equivalent to performing a prefix-sum [18] on the original regular histogram, and generating a 2D cumulative histogram can be realized using two scans on the original 2D regular histogram using both row-major order and transposed row-major order (to be detailed in Section IV), we have developed a simple yet effective parallel approach to derive $|B1_i|$ and $|B2_i|$ counts for all bins from the two sets of MBBs of the two input datasets in a spatial join and use them to choose the appropriate grid resolution for the spatial joins under GPU memory constraints. To the best of our knowledge, this is the first work on parallel selectivity estimation for spatial joins on GPUs considering resource constraints. Before we present the details of the parallel designs and implementations in Section IV, we next provide a framework for parallel spatial join processing on GPUs. This will not only put our proposed technique in context, but more importantly, it will help explain the role of selectivity estimation in the complete spatial join process, and subsequently, identify various research opportunities and associated technical challenges on spatial data management on GPUs.

III. PARALLEL SPATIAL JOIN FRAMEWORK ON GPUS

Spatial data is rich in data types and different spatial data types may allow different spatial operations, for example, distance calculation between points and polylines and point-in-polygon tests among points and polygons. We refer to the OGC SFS [10] for more details on spatial data modeling. While most of the existing spatial databases adopt Object-Relational data models for spatial data to extend relational database functionality to spatial data, extensive dynamic memory allocations to construct spatial objects in memory can cause significant overheads and is not cache friendly. To boost the performance of the in-memory data structures for complex and read-only spatial data, we have designed an array-based physical data layout scheme [17]. For complex spatial objects such as polylines and polygons, in addition to their vertex arrays, auxiliary index arrays are also created. These arrays can be efficiently streamed among disks, CPU memories and GPU memories. While we refer to [17] for a summary of our GPU-based techniques for spatial query processing, including grid-file, R-Tree and Quadtree based spatial indexing and spatial join techniques, in this study, we consider the general situation that spatial index structures are available for none of the input datasets involved in a spatial join and we thus resort to a simple grid-file based approach for spatial filtering as shown in the middle of Fig. 1, which requires a proper grid resolution to be chosen for performance as shown in the top-right part of Fig. 1.

While points can be easily grouped into grid cells by chaining sorting and reduction parallel primitives [18] (using grid cell identifiers as keys), MBBs of polylines and polygons may intersect with multiple grid cells. After both geometric objects are aligned to one or more grid cells, generating (*P*, *Q*) pairs can be transformed into a binary search problem. For each grid cell in the VPC vector, which stores the one-to-many mappings between the MBB of a geometric object in T1 to the grid cells that the MBB intersects, we search the cell in the VQC vector, which stores the one-to-many mappings between the MBB of a geometric object in T2 to the grid cells that the MBB intersects (center of Fig. 1). The matched objects in T1 and T2 will be paired for refinement. Clearly, for MBB pairs that cover multiple grid cells, the (*P*, *Q*) pairs will be duplicated and need to be removed to avoid redundant spatial refinements. All the steps can be realized using parallel primitives [18]. During the refinement phase, (*P*, *Q*) pairs will be assigned to GPU computing blocks for parallel processing as shown at the bottom part of Fig. 1.

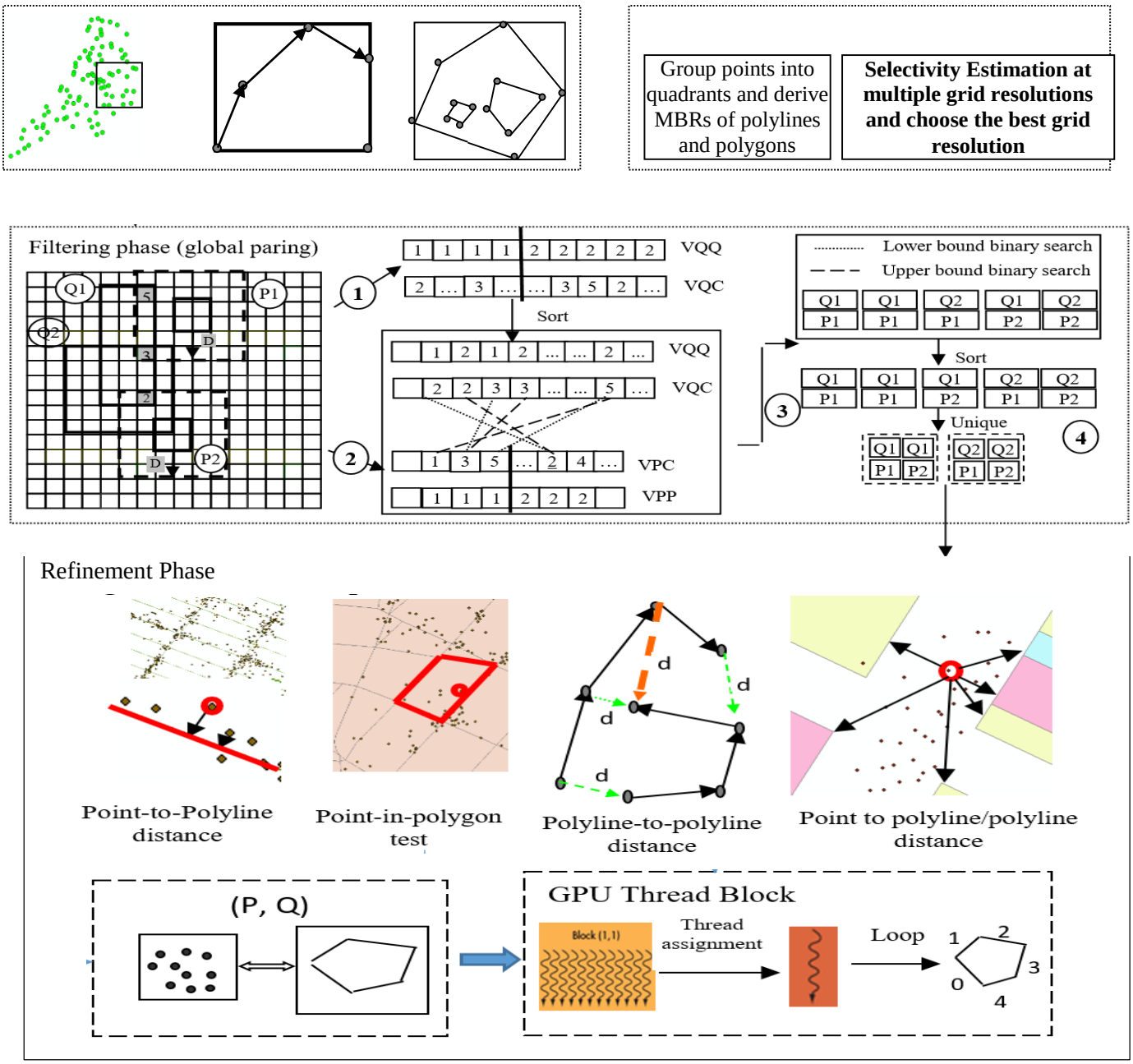

Fig. 1 A Framework of Parallel Spatial Join Processing on GPUs

As there will be multiple points/vertexes in both *P* and *Q* (here we treat grouped points as a point collection object), we assign one set of points/vertexes to threads in the computing block while looping through all the other sets of points/vertexes to derive results that will be associated with either points/vertexes or the pairs assigned to the computing block. This nested-loop style design is very efficient on GPUs as neighboring threads read neighboring points/vertexes in one object (assuming *P*) before a loop begins and write to neighboring positions for outputting results after the loop finishes while they access the same point/vertex in another object (assuming *Q*) throughout the looping process. The memory access pattern is perfectly coalesced which is critical for performance in GPU computing. As an example shown in the bottom part of Fig. 1, assuming that *P* contains *M* points in a grid cell and *Q* contains the *N* vertices of a polygon, we can

assign *M* points to threads in the GPU computing block while letting all threads loop through the *N* vertices. Depending on the sizes of points/vertexes in *P* and *Q* and the configurations of GPU computing blocks, we may need to reshape the O(*M***N*) computation to maximize the utilization of GPU hardware. For example, when *M* is less than the warp size (currently 32 on CUDA enabled GPUs [16]), we can loop through *K* (>=2) points in the *Q* polygon simultaneously and reduce the number of the looping steps to *ceiling*(*N*/*K*). Similarly, when *M* is larger than the number of threads in the computing block (assuming *T*), we may need to loop over the *M* points in *ceiling*(*M*/*T*) rounds. The parallel designs and implementations of point grouping, spatial filtering and spatial refinement are documented in more details in our previous works [4,5,6].

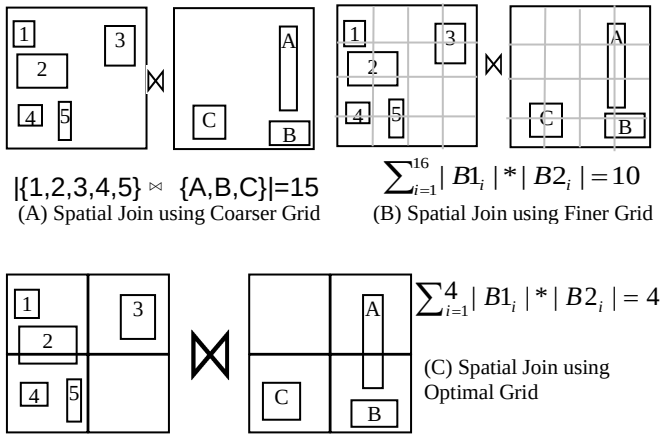

Fig. 2 Illustration of Choosing Optimal Grid Level in Minimizing Memory Footprint for Spatial Filtering

A major issue that we have encountered in applying the framework for practical spatial join applications is the difficulties in choosing grid cell resolutions which have a significant impact on GPU memory consumption in the filtering phase. When a large cell size is chosen, more MBBs from both input datasets will be associated with non-empty grid cells. As the design requires pairing all the MBBs from both datasets, very often a large number of pairs, i.e., $S = \sum_{i=1}^{N} |B1_i| * |B2_i|$ (where N is the number of non-empty cells and $|B1_i|$ and $|B2_i|$ are the numbers of MBBs associated with the non-empty cell *i*), need to be outputted before unique pairs can be computed and used in the refinement phase. Although the number of unique pairs might be small, the number of intermediate pairs can be too large to fit GPU memory (currently limited to a few GBs). On the other hand, if a small cell size is chosen, while $|B1_i|$ and $|B2_i|$ are likely to be smaller, *N* usually grows quadratically which may also incur a large number of intermediate pairs. For the example shown in Fig. 2, the grid on the right (Fig. 2C) is most memory efficient where the number of candidate pairs (4) is significantly smaller than that incurred when using a coarser grid (Fig. 2A) or finer grid (Fig. 2B). As such, it is desirable to choose a grid resolution that can minimize memory footprint.

IV. GPU PARALLEL SELECTIVITY ESTIMATION

Our parallel selectivity estimation technique is designed to compute $|B1_i|$ and $|B2_i|$ values efficiently on GPUs so that *S* can be computed on GPUs by a parallel reduction (prefix sum) [18]. Subsequently this requires computing the number of MBBs in each bin of both T1 and T2. An approach that utilizes cumulative histograms for this purpose has been proposed in [8] and illustrated in Fig. 3. However, we are not aware of previous work on parallelization of deriving 2D cumulative histograms and computing selectivity on GPUs. By treating the numbers of MBB corners that fall within grid cells as pixel values, we transform the problem of deriving 2D cumulative histograms from a set of MBBs to the problem of computing Summed-Area-Tables from image pixels [9]. We next present the details of our GPU-based parallel selectivity estimation technique.

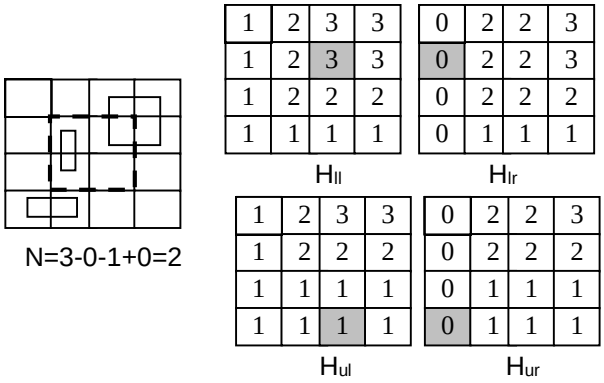

Fig. 3 Illustration of Spatial Cumulative Histogramming and Selectivity Estimation

The complete algorithm for our selectivity estimation technique is shown in Fig. 4. As shown in the top-left part of Fig. 4, the main algorithm computes the number of MBBs that intersect with each and every grid cell for the two input datasets (M1 and M2) separately, and the total number of possible MBB intersections across all grid cells (i.e., $S = \sum_{i=1}^{N} |B1_i| * |B2_i|$) can be computed using a *Reduce* parallel primitive [18], which is well understood in parallel computing and efficiently implemented in the CUDA Thrust library. The algorithm *compute_counts*, shown in the top-right part of Fig. 4, first extracts points of the four corners of the input MBB dataset and calls the algorithm *point_aggregation* to compute the numbers of points that fall into the grid cells with the desired grid resolution *r*. The algorithm *compute_counts* subsequently calls the algorithm *gen_sat* to generate the corresponding cumulative histogram array. This is done for all the four corners of MBBs and the results are stored in the four cumulative histogram arrays (H_{ll}, H_{lr}, H_{ul} and H_{ur}), respectively. Finally, the algorithm computes the number of MBBs that intersect with a grid cell for all grid cells based on the four arrays, i.e., $N^i = H_{ll}(x_2,y_2) - H_{lr}(x_1-1,y_2) - H_{ul}(x_2,y_1-1) + H_{ur}(x_1-1,y_1-1)$ by setting *x1=x2=c* and *y1=y2=r* where (*r*,*c*) is the row and column of the grid cell being processed.

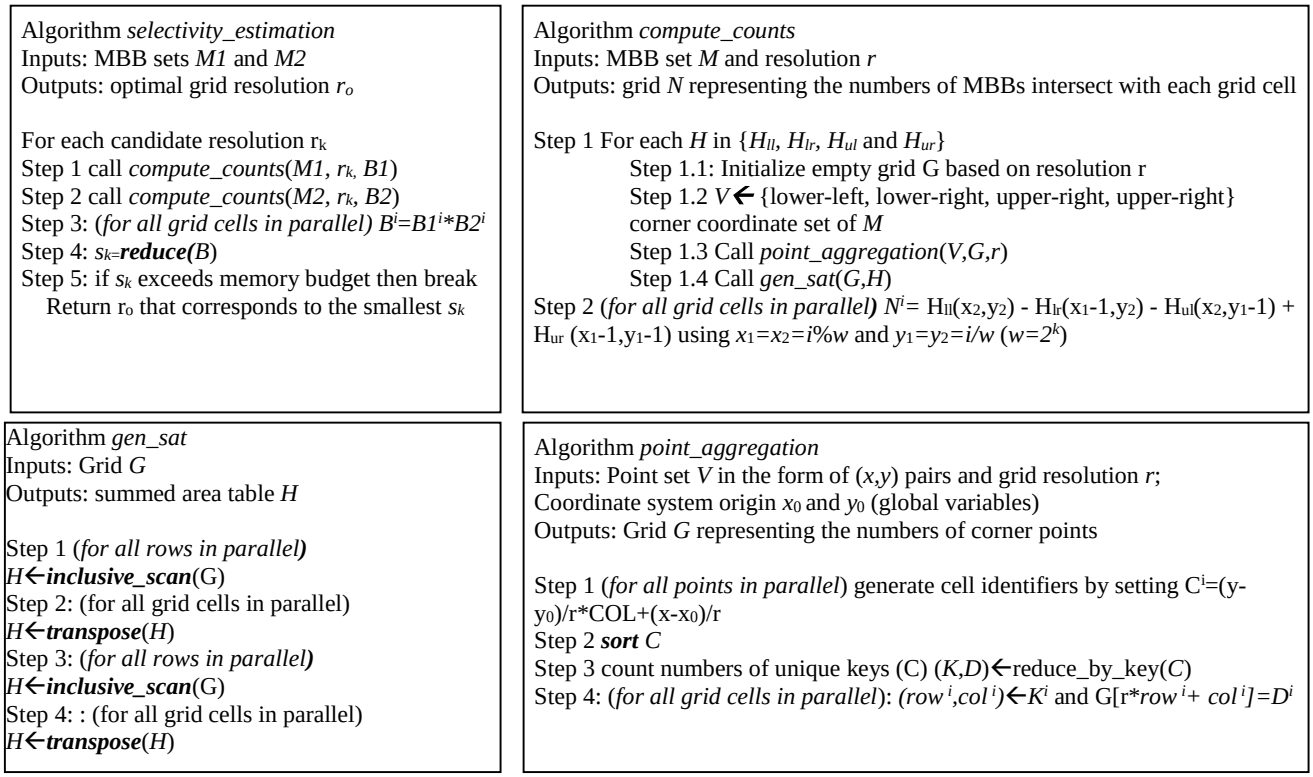

Fig. 4 Algorithms for Parallel Selectivity Estimation

The algorithm *gen_sat* requires more explanations on its parallel design and implementation. In the Summed Area Table algorithm [9], the corresponding array (cumulative histogram) can be realized by combining the row-wise and column-wise prefix sums and two matrix transpose operations. *Prefix Sum* can be directly realized using an *inclusive_scan* primitive provided by the CUDA Thrust library whose efficient implementations on GPUs have been extensively investigated. The *Transpose* operation can be either implemented using Thrust API by applying an element-wise functor to the *Transform* parallel primitive to copy an element to its corresponding transposed position in a 2D array, or can be natively implemented using CUDA to achieve better performance, which is also well studied. The algorithm *point_aggregation* chains three parallel primitives: *Transform* (to convert point coordinates to cell identifies), (radix) *sort* (based on cell identifiers) and *reduce_by_key* (to count the numbers of points that fall within a grid cell). Similar to the *Transpose* operation, the last step in the algorithm *point_aggregation* can also be implemented by using a *scatter* parallel primitive [18] or using CUDA directly.

In this study, we have implemented the selectivity estimation algorithm using parallel primitives provided by the Thrust library. The row/column-wise inclusive scans, the transpose operation and the element-wise operations that require a transform primitive are subsequently replaced with a native CUDA implementation for optimization purposes. We find that the optimized implementation is several times faster than the parallel primitives based implementation, largely due to avoiding library overheads and efficiently using thread-block level shared memory. The performance evaluation to be reported next is based on the optimized implementation.

V. Experiments and Discsussions

To validate our technique, we use two real datasets in our experiments. The first dataset contains 168 million taxi trip records each with a pick-up and drop-off location [42]. We generate quadrants for the pick-up locations from the point dataset by setting the maximum number of point in each quadrant to K=256 points (see [4] for details) and use the MBBs of the points in the quadrants. We call the first MBB set as Taxi and the number of MBBs in the set |Taxi|=1,746,795. The second dataset to participate in the spatial join test is the NYC MapPluto tax lot data with 735,488 polygons and 4,698,986 vertices [43]. We use the MBBs of the polygons as our second MBB set, i.e., |Pluto|= 735,488. We use four grid resolutions and vary grid sizes from $2^{10}*2^{10}$=1024*1024 to $2^{13}*2^{13}$=8192*8192 for selectivity estimation. All experiments are performed on a 2013 Nvidia GTX Titan GPU device with 2,688 cores and 6 GB memory.

Table 1 lists the computed numbers of pairs s_k and selectivity estimation runtimes in milliseconds at the four grid resolutions. Assuming that the chance of picking all grid resolutions for spatial filtering is the same, then the expected

number of estimated pair is AvgN=$\Sigma N_i/4$. After applying the selectivity estimation algorithm, we are able to pick up the grid resolution that has the minimum number of pairs minN=min(N_i) and thus the benefit is (AvgN-minN)/AvgN=34.8%. The total cost of the optimization is simply the total runtimes ΣTs_i=321 ms. The result indicates that it is possible to reduce the memory footprint of the spatial join by 34.8% in less than 1/3 of a second on a low-cost commodity GPU device, which is desirable.

Table 1 Estimated Pairs and Runtimes of the Four Grids

Grid Level k	Grid Size	# of Estimated Pairs (N)	Runtime (Ts) (ms)
13	8192*8192	78328554	205
12	4096*4096	40414590	63
11	2048*2048	43121125	31
10	1024*1024	86103593	22

From Fig. 4 we can see that the total runtime of the selectivity estimation has three parts: aggregating corner points to grids, generating SAT, and computing join pairs. For point aggregation, the algorithm *point_aggregation* is called 8 times in total for the four corners in the two input datasets and the cost is proportional to the number of MBBs in general (Step 4 is proportional to grid size). Similarly, for generating SAT, the algorithm *gen_sat* is called 8 times and the cost is linear with respect to grid size (width*height). The cost to sum up join pairs is a one-time cost and is generally negligible compared to others.

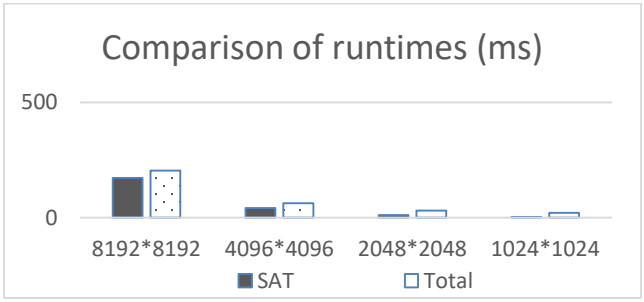

Fig. 5 Comparison between SAT and the Total Runtimes

In our experiment, point aggregation times are in the order of 10-15ms and 7-12ms for the two input datasets under the four grid sizes. To better understand how the runtime of generating SAT is affected by grid size, we have plotted the runtimes of both SAT generation and the total in Fig. 5. For small grid sizes (e.g., 1024*1024 and 2048*2048), the SAT generation time is only a small fraction of the total (3.85/22.19 and 11.71/30.67, respectively). However, for large grid sizes, the SAT generation time dominates (42.59/62.86 for 4096*4096 and 172.17/205.22 for 8192*8192, respectively). The results suggest that, although using coarse grids may miss the chance of finding an optimal grid resolution for spatial join, the low cost makes it desirable in many cases. In fact, for the particular spatial join in the experiment, using the grid level 11 (2048*2048) is only slightly worse (~6%) than using the optimal grid level 12 (4096*4096). When only the grid levels 10 and 11 are examined in the selectivity estimation, the runtime can be reduced from 321ms to 53ms.

We also note that, our current implementation handles the multiple grid levels independently, which can be further optimized. For the algorithm *point_aggregation*, it is possible to apply the multi-level grid-file based indexing technique that we have developed [6] to avoid redundant computation across multiple levels. For example, Step 1 of the algorithm computes cell identifiers for a given grid level (resolution) *r*. Instead of computing the cell identifiers from point coordinates, once the finest level cell identifiers are computed, the coarser level identifiers can be successively refined. This will also reduce the sorting cost in Step 2, which is the most expensive part in the algorithm. This is because refined cell identifiers are likely to be close to each other and expensive data movements in radix sort can be avoided. The optimization is left for our future work.

Although the experiments in this section only involve two datasets in a single spatial join, it is worthwhile to discuss the scenarios that a dataset involves in multiple (and possibly ad-hoc) spatial joins. From Fig. 4, it can be seen that the selectivity estimation algorithm actually allows processing the two input datasets independently except in Step 4 of the algorithm (the top-left part). While this may suggest that we can store B1 and B2 for reuse purposes and make the algorithm superfast as Step 4 typically only requires a runtime from a fraction to a few milliseconds, the cost to load the pre-computed SAT grids from disks may be overwhelming. For a 1024*1024 grid which requires 4MB storage, the disk loading time is already 40ms, assuming a typical 100MB/s disk I/O speed. The cost will increase to 64*40=2560ms for a 8192*8192 grid, which is dozens of times slower than on-demand computation based on our experiments. Although it is intuitive to use compression to reduce disk I/O time, SAT grids are dense arrays (see the examples in Fig. 3) and are likely to leave very little room for deep compression using conventional compression algorithms. We note, however, the grids right after aggregating MBB corner points are typically quite sparse for clustered real world data and can be easily compressed. We are in the process of applying our Bitplane Bitmap Quadtree (BQ-Tree) compression technique that is efficient on both GPUs and multi-core CPUs [44,45] for this purpose. The results will be reported in our future work.

VI. Conclusion and Future Work

In this study, we have provided a parallel selectivity estimation technique to reduce memory footprint in spatial join processing on GPUs where memory capacity is typically a limiting factor in processing large-scale data. Experiments on joining the two MBB sets with MBBs at the orders of millions have shown that our technique is able to reduce memory footprint by 38.4% in about 1/3 of a second when histograms are computed on-demand at multiple grid resolutions from 1024*1024 to 8192*8192. Preliminary results demonstrate that our technique is effective with a simple design and implementation.

For future work, first, we would like to incorporate the two optimization modules discussed above, i.e., successively generating multi-level grids for point aggregation and compression on grids after point aggregations to support reuse when a dataset is likely to be used in multiple spatial joins. Second, from a low-level code optimization perspective, there are opportunities to eliminate writing-out and reading-in intermediate results to/from GPU global memory which are expensive, i.e., kernel fusion using compilation terminology, and we would like to pursue this direction. Finally, for selectivity estimation on very large datasets and requiring high resolution grids, it is likely that GPU memory capacity will again be a limiting factor for the proposed selectivity estimation technique itself. We plan to adopt a multi-level approach to extend the technique by using CPU memory as a buffer and dynamically bringing blocks in CPU memory to GPU memory for block-wise selectivity estimation before the blocks are combined.

ACKNOWLEDGEMENT

This work is supported through NSF Grants IIS-1302423 and IIS-1302439.

REFERENCES

[1] https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/

[2] Mishra, P, and Margaret, E. H (1992). Join processing in relational databases. ACM Computing Surveys. 24(1) 63-113.

[3] Jacox, E. H. and Samet, H. (2007). Spatial join techniques. ACM Transaction on Database System 32(1).

[4] Zhang,J., You,S., Gruenwald,L. : Parallel online spatial and temporal aggregations on multicore CPUs and many-core GPUs Inf. Syst. 44: 134-154, 2014

[5] Zhang, J., You, S. and Gruenwald, L (2012). U2STRA: High-Performance Data Management of Ubiquitous Urban Sensing Trajectories on GPGPUs. Proceedings of ACM City Data Management Workshop (CDMW).

[6] Zhang, J. and You, S. (2012). Speeding up Large-Scale Point-in-Polygon Test Based Spatial Join on GPUs. Proceedings of ACM BigSpatial Workshop.

[7] Samet, H. (2005). Foundations of Multidimensional and Metric Data Structures Morgan Kaufmann.

[8] Jin, J., An, N. and Sivasubramaniam, A (2000). Analyzing range queries on spatial data. In Proc. IEEE ICDE'00.

[9] https://en.wikipedia.org/wiki/Summed_area_table

[10] http://www.opengeospatial.org/standards/sfs

[11] Aji, A., Wang,F., et al.(2013) HadoopGIS: A High Performance Spatial Data Warehousing System over Mapreduce Proc. VLDB Endow., vol. 6, no. 11, pp. 1009-1020.

[12] Eldawy,A., Mokbel, M F., Jonathan, C. (2016) HadoopViz: A MapReduce framework for extensible visualization of big spatial data Proc. ICDE'16, 601-612.

[13] Eldawy, A. and Mokbel M. F. (2016). The era of Big Spatial Data. In proc. IEEE ICDE, 1424-1427

[14] Appuswamy, R., Gkantsidis, C. et al (2013). Scale-up vs Scale-out for Hadoop: Time to Rethink? In Proc. ACM SOCC'13.

[15] Hennessy, J. L., Patterson,D. A. Computer Architecture: A Quantitative Approach (5th Ed.)Morgan Kaufmann, 2011

[16] Kirk, D. B., Hwu,W.-m. W.: Programming Massively Parallel Processors: A Hands-on Approach,2nd ed., Morgan Kaufmann, 2012.

[17] Zhang, J., You, S., Gruenwald, L. :Large-Scale Spatial Data Processing on GPUs and GPUAccelerated Clusters ACM SIGSPATIAL Special, vol. 6, no. 3, pp. 27-34, 2014.

[18] McCool, M., Robison, A.D. , Reinders, J.: Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufmann, 2012.

[19] Beigel,R. and Tanin, E. (1998). The Geometry of Browsing. Proceedings of LATIN'98: Theoretical Informatics, 331-340.

[20] Acharya,S., Poosala,V. and Ramaswamy,S. (1999). Selectivity estimation in spatial databases. Proceedings of SIGMOD, 13–24.

[21] Aboulnaga, A. and Naughton, J. F. (2000). Accurate estimation of the cost of spatial selections. Proceedings IEEE ICDE, 123-134.

[22] An, N., Yang, Z.-Y., and Sivasubramaniam, A. (2001). Selectivity estimation for spatial joins. Proceedings of IEEE ICDE, 368-375.

[23] Mamoulis, N. and Papadias, D. (2001). Selectivity Estimation of Complex Spatial Queries. Proceedings of SSTD, 155-174.

[24] Wang, M., Vitter, J., et al (2001). Wavelet-Based Cost Estimation for Spatial Queries. Proceedings of SSTD, 175-196.

[25] Choi, Y-J. and Chung, C-W. (2002). Selectivity estimation for spatio-temporal queries to moving objects. Proceedings of ACM SIGMOD, 440-451.

[26] Sun, C., Agrawal, D. and El Abbadi, A. (2002). Exploring spatial datasets with histograms. Proceedings of IEEE ICDE, 93-102

[27] Lin, X,. Liu, Q., et al. (2003). Multiscale histograms: summarizing topological relations in large spatial datasets. Proceedings of VLDB, 814-825.

[28] Belussi,A., Bertino, E. and Nucita A. (2004). Grid based methods for estimating spatial join selectivity. Proceedings of ACM-GIS, 92-100.

[29] Das, A., Gehrke, J., and Riedewald, M. (2004). Approximation techniques for spatial data. Proceedings of SIGMOD, 695-706.

[30] Zhang, D., and Tsotras, V. J. and Gunopulos, D (2004). Efficient aggregation over objects with extent. Proceedings of PODS, 121-132.

[31] Elmongui, H., Mokbel, M. et al. (2005). Spatio-temporal Histograms. Proceedings of SSTD, 19-36.

[32] Gunopulos, D., Kollios, G., et al (2005). Selectivity estimators for multidimensional range queries over real attributes. The VLDB Journal, 137-154.

[33] Sun, C., Bandi, N. et al (2006). Exploring spatial datasets with histograms. Distributed and Parallel Databases 20(1) 57-88.

[34] Sun, J., Tao, Y. et al (2006). Spatio-temporal join selectivity. Information Systems 31(8):793-813.

[35] Eavis,T. and Lopez,A. (2007). rK-Hhist: an R-tree based histogram for multi-dimensional selectivity estimation. In Proc. CIKM'07, 475–484.

[36] Luo, J., Zhou, X, et al (2007). Selectivity estimation by batch-query based histogram and parametric method. In Proc. ADC'07. 93-102.

[37] Huang, D-S., Heutte, L.and Loog, M (2007). Spatial Selectivity Estimation Using Cumulative Density Wavelet Histogram. In Proc. ICIC'07, 493-504.

[38] Roh, Y. J., Kim, J. H. et al. (2010). Hierarchically organized skew-tolerant histograms for geographic data objects. In Proc. SIGMOD'10, 627-638.

[39] Roh, Y-J., Kim, J-H, et al (2011). Efficient construction of histograms for multidimensional data using quad-trees. Decision Support Systems 52(1), 82 -94.

[40] Achakeev, D. and Seeger, B. (2012) A class of R-tree histograms for spatial databases. In Proc. ACM-GIS'12. 450-453.

[41] Mai, H., Kim, J. et al (2013). STHist-C: a highly accurate cluster-based histogram for two and three dimensional geographic data points. GeoInformatica 17(2) 325-352.

[42] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[43] http://www1.nyc.gov/site/planning/data-maps/open-data.page

[44] Zhang, J., You, S. and Gruenwald, L. (2011). Parallel quadtree coding of large-scale raster geospatial data on GPGPUs. In Proc. ACM-GIS' 11, 457-460

[45] Zhang, J., You, S. and Gruenwald, L. (2015). Quadtree-based lightweight data compression for large-scale geospatial rasters on multi-core CPUs. In Proc. IEEE BigData'15, 478-484.