Hierarchical Filter and Refinement System over Large Polygonal Datasets on CPU-GPU

Yiming Liu
Computer Science Dept.
Marquette University
yiming.liu@marquette.edu

Jie Yang
Computer Science Dept.
Marquette University
jie.yang@marquette.edu

Satish Puri
Computer Science Dept.
Marquette University
satish.puri@marquette.edu

***Abstract*—In this paper, we introduce our hierarchical filter and refinement technique that we have developed for parallel geometric intersection operations involving large polygons and polylines. The inputs are two layers of large polygonal datasets and the computations are spatial intersection on a pair of cross-layer polygons. These intersections are the compute-intensive spatial data analytic kernels in spatial join and map overlay computations. We have extended the classical filter and refine algorithms using PolySketch Filter to improve the performance of geospatial computations. In addition to filtering polygons by their Minimum Bounding Rectangle (MBR), our hierarchical approach explores further filtering using tiles (smaller MBRs) to increase the effectiveness of filtering and decrease the computational workload in the refinement phase.**

We have implemented this filter and refine system on CPU and GPU by using OpenMP and OpenACC. After using R-tree, on average, our filter technique can still discard 69% of polygon pairs which do not have segment intersection points. PolySketch filter reduces on average 99.77% of the workload of finding line segment intersections. PNP based task reduction and Striping algorithms filter out on average 95.84% of the workload of Point-in-Polygon tests. Our CPU-GPU system performs spatial join on two shapefiles, namely USA Water Bodies and USA Block Group Boundaries with 683K polygons in about 10 seconds using NVidia Titan V and Titan Xp GPU.

***Index Terms*—HPC, Parallel algorithm, OpenACC, OpenMP**

I. Introduction

In Geographic Information Systems (GIS) and spatial databases, vector geometries like polygons and polylines are used to represent real-world objects. The input to map overlay and spatial join queries are two layers of geo-spatial data. Spatial query operations often require expensive computational geometry algorithms. For computational efficiency, query operations are carried out in two phases. The first phase is a filter phase where complex geometries are approximated by their minimum bounding rectangle (MBR). Filter phase employs spatial data structures like R-tree built using MBRs of geometries. Working with MBR representation is faster compared to actual geometries that may contain thousands of vertices. Geometries that could not possibly satisfy the query condition are removed. The output of filter phase consists of candidate pairs that may or may not be part of the final output. The drawback of filtering using MBR is that it produces many false hits because of MBR approximation. As such, in the second phase of refinement, actual geometries are used to produce correct results by detecting and removing false hits.

With two layers R and S as inputs, the output of spatial join is a collection of pairs (r, s) such that $r \in R$, $s \in S$, and r and s overlap spatially. An example of spatial join query is "Find all the places where roads cross rivers". Overlap is one of the spatial relations between r and s. Other spatial relations are *Intersects*, *Touches*, *Contains*, etc. Spatial databases and GIS support such topological relations on two layers of geometries.

In this paper, we extend the classical filter and refine strategy by introducing $PolySketch$ technique. The basic idea is to represent a large geometry using its sketch that is made up of a collection of tiles. Each tile is a subset of contiguous vertices with the corresponding MBR induced by the subset. Our $PolySketch$ filter not only reduces the candidate pairs but also reduces the workload in the refinement phase.

Refinement phase involves computational geometry algorithms on a variety of shapes. Computing the topological relations and geographic map overlay requires two kernels namely, line segment intersection (LSI) and Point-in-Polygon (PNP). An example of polygon intersection is shown in Figure 1. There are two overlapping polygons P1 and Q1. The first step is to find line segment intersection vertices (black) among line segments from the two polygons as shown in Figure 1(a). The second step requires PNP function to find polygon vertices that are inside another polygon e.g., one vertex (red) of Q1 is inside P1 and two vertices (blue) of P1 are inside Q1 as shown in Figure 1(b). Finally, output polygon(s) are produced by combining the output of LSI and PNP functions. An output polygon is shown in Figure 1(c).

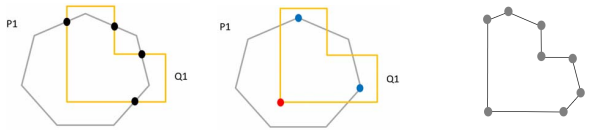


Fig. 1. (a) Line segment intersection vertices, (b) Vertices inside another polygon, and (c) Output polygon (Best Viewed in Color)

It has been shown that as geometries are getting larger in size, the refinement phase is taking most of the time [1]. Decreasing the number of candidates produced in the filter phase also reduces the workload in the refinement phase. Therefore,

2640-0316/19/$31.00 ©2019 IEEE
DOI 10.1109/HiPC.2019.00027

we propose applying a hierarchy of MBR and $PolySketch$ filter to improve the filter efficiency. Not all segments of a polygon will intersect with the segments of another polygon. Expensive polygon intersections in the refinement phase can be possibly eliminated by using the sketch of a polygon. Further improvement is possible by the GPU-acceleration of computational geometry algorithms in the refinement phase.

We illustrate the benefit of our new approach using an example that shows how hierarchical filtering reduces the overall workload. Let us consider we have two layers of polygons L1 and L2. L1 and L2 consist of 310 and 500 polygons respectively. Table I shows nine candidate pairs and each pair has two polygons (P and Q) from L1 and L2 whose MBRs overlap with each other. The number of vertices in a polygon is also shown below the polygon ID. As shown in Table II, the application of $PolySketch$ Filter eliminates some line segments that can be safely ignored from further refinement. Moreover, in case of polygons from a few of the candidates, MBR-based Filter eliminates all the line segments. As a result, we can discard the pairs that have zero line segments after applying hierarchical filtering. Since, polygon intersection with n and m line segments is an $O(n * m)$ time algorithm, reducing the line segments by filtering is beneficial.

TABLE I
CANDIDATE PAIRS BEFORE HIERARCHICAL FILTERING

	1	2	3	4	5	6	7	8	9
L1	P3	P3	P21	P24	P88	P88	P99	P236	P300
	35	35	199	652	998	998	152	4652	52
L2	Q5	Q7	Q56	Q3	Q5	Q12	Q5	Q5	Q457
	65	22	659	832	65	529	65	65	1526

TABLE II
ELIMINATING LINE SEGMENTS BY HIERARCHICAL FILTERING

	1	2	3	4	5	6	7	8	9
L1	P3	P3	P21	P24	P88	P88	P93	P236	P427
	35	0	156	0	0	451	112	324	52
L2	Q5	Q7	Q56	Q3	Q5	Q12	Q5	Q5	Q637
	0	0	34	0	65	256	32	30	0

The contributions of this paper are as follows:

1) We have introduced a new filtering technique based on PolySketch concept which can speedup LSI function used by spatial join and map overlay computations involving large polygons and polylines.
2) Our OpenACC-based GPU implementation performs spatial join on two shapefiles, namely USA Water Bodies and USA Block Boundaries in about 10 seconds using NVidia Titan V and Titan Xp GPU.

The rest of the paper is organized as follows. Section II provides the background and related work. Section III introduces $PolySketch$ Filter. Section IV provides design and algorithm details on our hierarchical filter and refine system. The experimental results are discussed in Section V. Finally, we conclude in Section VI.

II. BACKGROUND AND RELATED WORK

Plane-sweep is a fundamental technique in computational geometry and it has been parallelized on multi-core and manycore architectures [2]–[4]. Boolean set operations like union and intersection on polygons require line segment intersections and point-in-polygon test [5]. GPU-based acceleration of segment intersections and point-in-polygon test have been studied in the domain of GIS and spatial databases [1], [6]–[9].

We have designed theoretical PRAM algorithms and multi-threaded implementations for polygon clipping [10], [11]. We used the intersection of two cross-layer polygonal MBRs (CMBR) earlier in our GPU-based spatial join system called GCMF [12] to filter out candidate pairs that do not need further refinement. CMBR is effective in cases where it can filter out the majority of the line segments. This leads to a reduction in workload. It was observed that in some cases CMBR was not effective in workload reduction. So, CMBR technique was further improved by creating grid inside the area of CMBR for further filtering [13]. As opposed to CMBR, $PolySketch$ is a hierarchical technique. However, our $PolySketch$ system can employ CMBR technique in a hierarchical manner to weed out the pairs of cross-layer tiles that do not need further refinement. Other approaches used in literature include $PixelBox$ where geometries represented as co-ordinates are converted to raster format (pixels) to leverage image processing using a GPU [14].

Spatial partitioning of geometries using techniques like uniform grid [9], quadtree and binary space partitioning has been studied in literature. In our system, we do not explicitly do spatial partitioning. Instead, we use data partitioning by tiling. Tiling induces spatial partitioning. We do not use uniform or adaptive grid partitioning of input layers of spatial data.

Different methods employed in filter and refine based spatial query processing have been discussed in [15]. R-tree is used for building a spatial index and MBR query [16]. Partitioning a polygon by decomposing its area into smaller and simpler geometries has been studied earlier [17]. Existing tools in GIS invoke computational geometry algorithms on shapes made of 2D co-ordinate data. In computer graphics, a complicated shape is often decomposed by triangulation. For efficiency, triangulations are used instead of actual geometry. In contrast, $PolySketch$ is a recursive tiling of polygon boundary only; the interior of a polygon is disregarded altogether. As such, sketch of a polygon does not work in the same way as a polygonal MBR or classical polygon partitioning because $PolySketch$ approximates the boundary only. In short, $PolySketch$ is not a replacement for MBR in classical filter and refine scenario. It must be used in conjunction with point-in-polygon test (PNP) to implement standard $intersects$ predicate in spatial join.

In addition to representing geometries using MBRs, convex and concave hull can also be used. Convex hull can be used as a replacement for MBR in spatial queries. Creating a PolySketch is a linear time operation, on the other hand, convex hull algorithms require $O(n\log n)$ time. Douglas-

Peucker line simplification algorithm and its variants reduce the number of points to represent a curve [18]. Line simplification is primarily used for visualization on a map. In case of geometries like polygons, using the simplified version may not yield correct results for spatial queries. $PolySketch$ is similar to hierarchical bounding technique used for line-curve intersection and curve-curve intersection [19].

We have used compiler directives for GPU-based parallelization of computational geometry algorithms earlier [4]. In our OpenACC-based spatial join implementation, we used classical filter and refine technique where filter phase was done on a CPU using R-tree query and refinement phase was done on a GPU. The performance for large data was unsatisfactory when compared to sequential implementation of spatial join on a CPU. This basically motivates the present work.

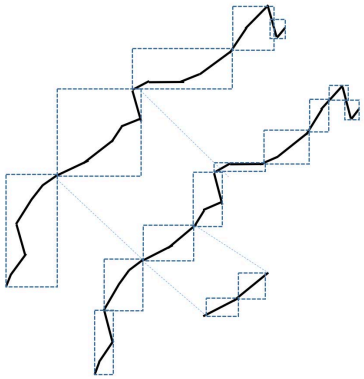


Fig. 2. PolySketch of polyline data composed of tiling at three different levels.

III. PolySketch and CMBR Filter

Sketch of a polygon/polyline is made by tiling its boundary in such a way that one tile represents an MBR of the vertices in that tile. These tiles are contiguous and two adjacent tiles share a vertex. A sketch is designed as a lightweight representation to be used in the filter phase of a filter-and-refine algorithm in a spatial computation, e.g., join, overlays. Figure 2 shows the hierarchical tiling approach. A tile is defined as a C structure. When compared to MBR of a geometry, a sketch of a geometry has less dead-space. As such, better filter efficiency is possible at the cost of additional space requirement.

Listing 1. Tile Data Structure

```
struct Tile {
  int start, end;
  vertex *v; // vertex array
  MBR mbr; //MBR of v[start] to v[end]
};
```

A PolySketch for a geometry with n vertices and tile length set as b consists of $\lceil \frac{n}{b} \rceil$ tiles. Since, in each tile, an MBR for the vertices in that tile needs to be computed, building a sketch of a polygon is $O(n)$ operation. An MBR of an entire geometry is also $O(n)$ operation. As will see later in the experimental results, sketching provides a space-time tradeoff because of its hierarchical nature.

A tree can be constructed to represent a hierarchy of PolySketches at different levels. Using the leaf-level tiles, internal nodes of the tree can be constructed using union of two successive tiles. An MBR of a polygon can be thought of as its level 0 sketch with its start index as 0 and end index as the number of vertices in the polygon. For a polygon with N vertices, there are O(log N) sketches possible in a tree-based representation. However, it suffices to use a few levels only as shown in Figure 2 for space-efficiency.

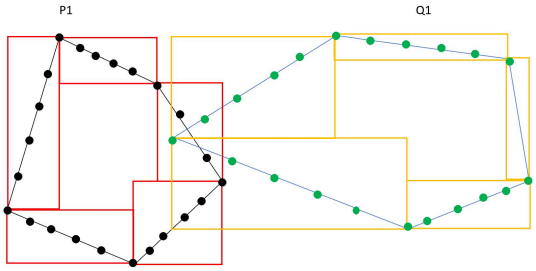
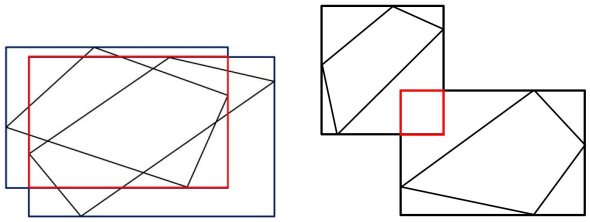


Fig. 3. Polygon intersection using PolySketch. Tiles for each polygon is shown in different colors.

As shown in Figure 3, there are two polygons P1 from layer1 and Q1 from layer 2. The tile-size is set as six line segments. Q1 consists of twenty-one line segments, so it is divided into five tiles. For polygon intersection between P1 and Q1, we first check if their corresponding tile-MBRs overlap or not. If some tile-MBRs from P1 and Q1 overlap, we record those tile pairs and use LSI function for those pairs. If there is no tile-MBR overlap, we discard this task for LSI function. In Figure 3, we can see that there are three pairs of tiles that have overlap. A tile located in the lower left corner of Q1 overlaps with two tiles located in the right-side corner of P1. Similarly, another tile located in the upper left corner of Q1 overlaps with one tile located in the upper right corner of P1. Other tiles and their corresponding vertices can be safely ignored in LSI function.

Checking if two tile-MBRs overlap is computationally cheaper than finding the segment intersection between two line segments. In one of the datasets that we have used, about 13% of polygons have more than 500 vertices. Since, a tile's MBR contains a fraction of the vertices of a polygon, using it in place of actual vertices in the filter phase is a cost-effective strategy.

CMBR Filter: Common Minimum Bounding Rectangle (CMBR) is a method which is based on Minimum Bounding Rectangle (MBR) of a polygon. For a pair of polygons, the CMBR is an area where the two MBRs overlap. Figure 4 shows two examples of CMBR. Black rectangles are two polygons' MBRs and the red rectangle is their CMBR. When two MBRs overlap as shown in Figure 4(b), it is possible that these two polygons do not overlap. Such cases can be safely ignored in the refinement phase. This particular case can be detected in the filter phase itself by checking if the vertices



(a) (b)

Fig. 4. Common minimum bounding rectangle examples (Red rectangle is CMBR; Best viewed in color.)

of a polygon lie in the CMBR area or not. Line segments that do not occupy the CMBR area can be safely ignored for LSI function. CMBR of a polygon has been used in earlier work [12] to improve the performance of spatial join on GPU.

In some cases, common MBR area may contain most of the vertices of the overlapping polygons. This particular case is shown in Figure 4(a). This limitation of CMBR approach makes it less effective [13]. A filter based on $PolySketch$ provides an alternative way of reducing the workload in LSI function. In general, there are not many segment intersections in polygon overlay [11], so even when the CMBR is less effective, $PolySketch$ can provide significant workload reduction. In addition, the strengths of CMBR and $PolySketch$ can be combined to get better performance.

In spatial join, the output polygon produced by polygon intersection is not required. Spatial join is based on a boolean predicate, e.g., $Intersects$. $Intersects$ predicate returns true if two polygons have line segment intersection or a polygon is inside another polygon. In polygon overlay, output polygon needs to be computed as well. This requires finding all line segment intersections as well as vertices of the polygon that are inside another polygon. Because of this difference, polygon overlay is computationally more expensive than spatial join. In our work, we compute all the segment intersections and the vertices of a polygon that are inside another one. The number of segment intersections (can be quadratic in the worst case) is variable for each candidate pair, so handling it on GPU either requires redundant computation because of counting the number of intersections a priori or using atomic locks while storing them.

IV. Hierarchical Filter and Refinement System

When two layers of geometries are overlaid or superimposed in a geographic map, there can be millions of candidate pairs whose MBRs have overlap and need further refinement using computational geometry algorithms. Even though there are PRAM based parallel algorithms available in literature [10], [11], optimal $O(n\log n)$ algorithms for geometric intersection are not available on GPUs. On CPUs, sequential plane-sweep based algorithms are used. For practical parallel implementations, naive $O(n^2)$ algorithms or grid partitioning are used [9], [14]. Even on massively parallel hardware, the quadratic runtime of the naive algorithms results in unacceptable high latency [4]. Grid partitioning may not handle skewed data efficiently. Moreover, partitioning polygons in a uniform or adaptive grid has the disadvantage of a polygon spanning multiple grid cells, thereby increasing redundancy due to duplication of geometries across grid lines.

For the original data, we do not use spatial partitioning using grids. Our approach is to use a combination of filter techniques to reduce the workload in the refinement phase. The input to CMBR and $PolySketch$ Filters is the list of candidate pairs produced by querying R-tree data structure built using the MBRs of the input geometries. We refer to these candidate pairs as tasks because we process them concurrently on a GPU.

Algorithm 1 Hierarchical Filter-based Segment Intersection

1: Input: Two polygon layers L1 and L2
2: Build R-tree using MBRs of polygons from L1
3: tasks ← Rtree Query using MBRs of polygons from L2
4: newTasks ← Apply CMBR Filter on tasks
5: Apply PolySketch Filter for each newTask
6: **Refine**: Line Segment Intersection Function

A. System Design Overview

In this section, we present how our hierarchical filter and refinement system works. Given two layers of polygons, Algorithm 1 shows the order of application of different filters to find cross-layer line segment intersections in the refine phase. At first, we check which polygon's MBR overlaps with others by using R-tree. If some MBRs overlap, we store these polygon pairs as tasks (T) and every task has two polygons. Then we use CMBR Based Task Reduction Algorithm to check which tasks are valid tasks (T1) or invalid tasks (T2) for LSI function. After this, we fix the tile-size for creating tiles for the valid tasks (T1) and use $PolySketch$ Filter to reduce the number of line segments and workload. In the last step, we use LSI function to detect the tasks where two polygons have line segment intersection(s).

PNP operation for Polygon: There are some cases when a polygon is contained inside another polygon completely. In spatial database, these cases belong to the $within$ or $contains$ spatial relations. Finding if a point is inside another polygon is $O(n)$ operation because all the n segments of the polygon need to be examined. As such, a brute-force check for an entire polygon is quadratic in the number of vertices of a polygon. When a polygon A is contained inside another polygon B, then MBR of A is also contained inside MBR of B. However, the reverse is not always true. Our algorithm for PNP can detect these $contains$ relation without resorting to a quadratic algorithm. Our algorithm can also detect those cases where MBR of A is contained inside MBR of B, but A and B are disjoint polygons. As a result, our system can safely ignore these tasks from expensive refinement operation later. Identifying these cases correctly is possible because we take advantage of the CMBR filter for optimizing PNP operations as well. More details for PNP-based Task Reduction Algorithm are in Subsection IV-D.

Now we will discuss the hierarchy of filters that our system employs to reduce the number of tasks and overall workload.

B. CMBR Filter Based Task Reduction

The input to this filter is the list of candidate pairs generated by R-tree queries. Each candidate consists of a pair of MBR-overlapping geometries. However, by virtue of CMBR Filter [12], those cross-layer pairs of polygons whose Minimum Bounding Rectangles (MBRs) intersect but their rectangular intersection does not contain line segments from both layers can be safely ignored because those pairs will not have segment intersections. We call such tasks invalid tasks and save computation time by discarding them from further processing in LSI function. An example of an invalid task is shown in Figure 4(b).

Our system classifies the pairs that have line segments inside or across CMBR from two cross-layer polygons as valid tasks; both polygons have segments that are contained inside or across the CMBR boundary. We store this task because the polygons can potentially intersect and need further refinement. An example of a valid task is shown in Figure 4(a).

CMBR Filter works well in eliminating some tasks from further refinement. In Subsection V-C, we compare the execution time performance and filter efficiency of using $PolySketch$ vs CMBR w.r.t. LSI function.

C. Workload Reduction by PolySketch Filter

As shown in Figure 3, a polygon can be represented as a collection of tiles. The tile-size is user-defined. Intersection of two polygons can be expressed as intersection of their PolySketches. Since, tile-MBR in a $PolySketch$ captures the actual area covered by line segments in that tile, line segment intersection can be carried out in two phases: 1) filter phase where tile-MBRs are used for intersection test and 2) refine phase where we only consider the line segments from those tiles that have overlap in LSI function. This is the essence of $PolySketch$ Filter.

In CMBR Filter, we need to compare all segments inside CMBR of one polygon with all segments inside CMBR of another polygon. If CMBR is large as shown in Figure 4(a), we cannot decrease a lot of segments from both polygons which affects workload in LSI function. However, by using $PolySketch$, a line segment in tile A needs to be compared against the segments of only those tiles which overlap with tile A. Algorithm 2 shows how to apply $PolySketch$ Filter and Refine for polygon intersection tasks using compiler directives supported by OpenACC.

There are certain scenarios where two polygons overlap but it is not detected by LSI function. Therefore, we use PNP function for further filtering of tasks.

D. PNP Based Task Reduction Algorithm

In this algorithm, we find out the vertices of a polygon that are contained inside another polygon. This is required to construct the output polygon for each task. We also find out those tasks where one polygon is entirely inside another polygon in an optimized way. These tasks result in valid output pairs for spatial join or polygon overlay operation. We also want to discard those tasks where polygons are disjoint so that we do not have to invoke quadratic PNP tests for an entire polygon. Figure 5 shows the flow chart of this algorithm.

Algorithm 2 Segment Intersections using PolySketch Filter

```
1:  #pragma acc data copyin(layer1Polygons, layer2Polygons)
    copyout(line segment intersections)
2:  #pragma acc parallel
3:  #pragma acc loop
4:  for each taskID ∈ taskArray do
5:      get polygon pair (p,q) using taskID
6:      #pragma acc loop
7:      for each tile t_p ∈ p.tiles do
8:          Calculate t_p.MBR
9:      end for
10:     #pragma acc loop
11:     for each tile t_q∈ q.tiles do
12:         Calculate t_q.MBR
13:     end for
14:     #pragma acc loop reduction (numSegIntersections)
15:     for each tile t_p ∈ p.tiles do
16:         #pragma acc loop
17:         for each tile t_q ∈ q.tiles do
18:             if (t_p.MBR overlaps t_q.MBR) then
19:                 Call LSI(t_p.segments, t_q.segments)
20:                 #pragma acc atomic
21:                 store segment intersections
22:             end if
23:         end for
24:     end for
25: end for
```

PNP based algorithm is used after the LSI Function. The intention of this algorithm is to discard invalid tasks for PNP function, divide valid tasks into two different types, do some pre-processing steps and use appropriate PNP functions for them. Before we use LSI Function, we use CMBR Filter to preprocess data to discard some invalid tasks for LSI function. However, we cannot discard these tasks for PNP function. We need to check all tasks (T) for PNP function.

Based on the output of CMBR Filter results, we classify the overall tasks so that we can treat them differently in order to reduce the PNP computation time. We divide all tasks (T) into three different types of tasks (S1, S2 and S3). S1 includes the tasks where two polygons have line segment intersections. S2 includes the tasks where two polygons do not have line segment intersection and guaranteed not to intersect. S3 includes the tasks where two polygons do not have line segment intersection but one polygon may be inside another polygon. Then, we use Striping Algorithm to preprocess data for the tasks in S1 which will be used in Stripe-based PNP function. Furthermore, we use constant vertex PNP function only for the tasks in S3 category. We discard the tasks in S2 category. This helps in reducing a lot of PNP workload. For

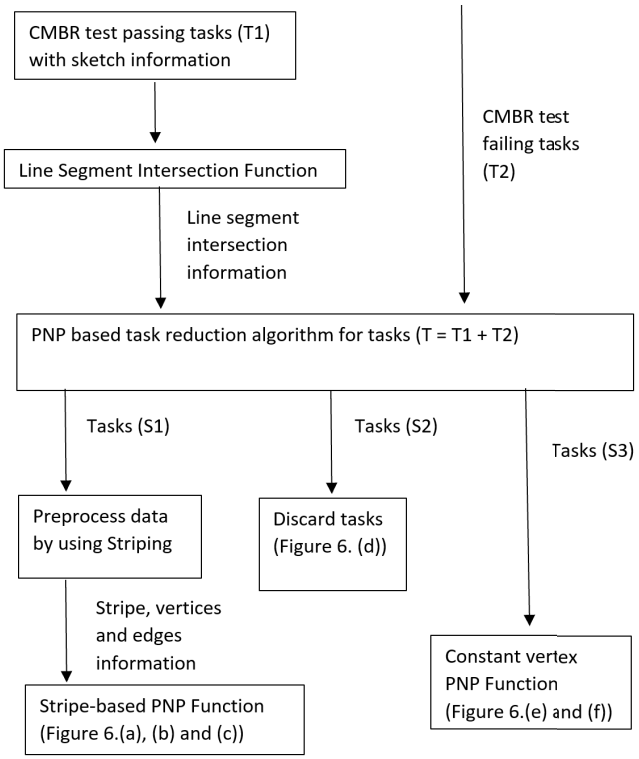


Fig. 5. Classifying PNP tasks after CMBR Filter

implementing filter and refinement steps, we use OpenMP and OpenACC to parallelize them on CPU-GPU system.

Valid tasks for Stripe-based PNP function: If two polygons have line segment intersection(s) (e.g. Figure 6(a)), we store the pair for Striping algorithm and Stripe-based PNP Function. Moreover, we also store special cases as shown in Figure 6(b) and Figure 6(c) for further processing.

Valid tasks for constant vertex PNP function: If two polygons do not have any line segment intersection, we check their MBRs. If one MBR of a polygon is inside another MBR, we store this task for constant vertex PNP function. The reason is that the smaller polygon may be totally inside the larger polygon when they do not have line segment intersection. In other words, all vertices of the smaller polygon may be inside the larger polygon. We also store which MBR includes another MBR because we only need to check whether the smaller polygon is inside the larger polygon. In addition, we do not need to check all vertices of the smaller polygon. It suffices to check a few vertices of smaller polygon whether they are inside or outside the larger polygon. Then, we know the smaller polygon is inside or outside the polygon. For illustration, Figure 6(e) and Figure 6(f) are two examples where we invoke PNP function for only a few vertices. In the experiments, we consider the output of PNP test for any five contiguous vertices to handle this special case.

Invalid tasks for PNP function: If two polygons intersect, there are two cases - a) there are line segment intersection(s)

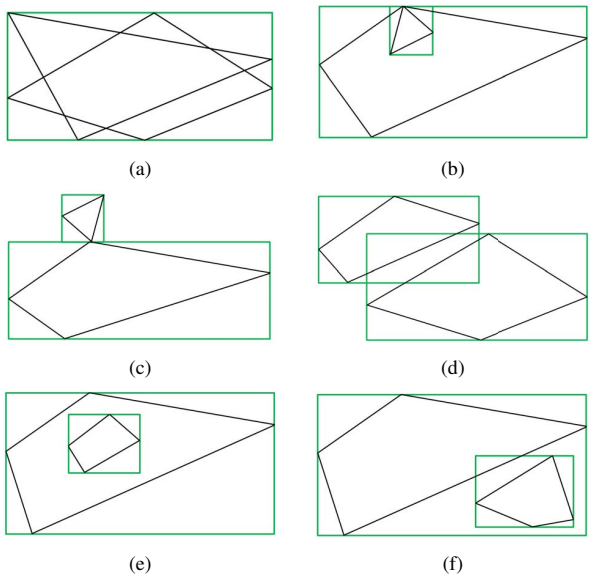


Fig. 6. PNP cases: (a) Two polygons have line segment intersections, (b) and (c) Two polygons *touch* each other, (d) Two polygons' MBRs overlap but there is no actual intersection, (e) One polygon is inside another polygon but there is no line segment intersection, (f) The smaller polygon is not inside another polygon but the smaller MBR is inside another MBR.

and b) there is no line segment intersection but one polygon is totally inside another polygon. Therefore, if two polygons do not have line segment intersection and no polygon's MBR is inside another polygon's MBR, they do not intersect. As such, we will discard this task. The reason is that if one polygon is inside another polygon, its MBR should be also inside another polygon's MBR. As shown in Figure 6(d), two polygons do not have line segment intersection and no MBR of a polygon is inside another MBR, so they cannot intersect.

E. Striping algorithm and Stripe-based PNP function

We have used striping to speedup PNP tasks. Striping is a filter technique used to optimize PNP function. Once the segments of the polygon are partitioned into stripes, PNP test for a vertex needs to consider only the line segments contained in or crossing a stripe. This reduces the workload for PNP function.

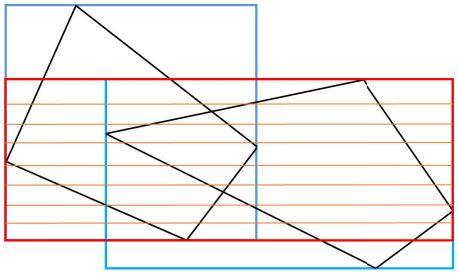


Fig. 7. An example of striping for Stripe-based PNP function

The area divided into stripes is the red rectangle as shown in Figure 7. We divide the area occupied by the red rectangle

into 8 cells. Striping algorithm considers all the line segments belonging to a task and maps a line segment to the cells where there is an overlap. In case a line segment overlaps with two or more cells, the segment is replicated in those cells. This is carried out by comparing y co-ordinates of vertices of a line segment with the cell boundaries. For Stripe-based PNP function, we need to check the vertices inside a cell only with another polygon's line segments inside or across the same cell. Therefore, the vertices need to be compared only with those line segments which overlap with the same cell. For GIS datasets with large polygons, this can potentially reduce a lot of workload.

Multi-GPUs: After the geometries have been partitioned into multiple cells, parallel processing of PNP tests can be carried out over multiple GPUs. As shown in Figure 7, there is no dependency among those eight cells. In order to utilize four GPUs, we can assign two cells to each GPU in a round-robin fashion. In our experiments, we have leveraged multiple GPUs to distribute PNP-based computations.

V. Experimental Results

A. Datasets

We have used three datasets to evaluate our system: (1) Urban, (2) Water, and (3) Lakes. The details are shown in Table III. Urban and Water are from http://www.naturalearthdata.com and http://resources.arcgis.com. The third dataset (Lakes) is from http://spatialhadoop.cs.umn.edu/datasets.html.

TABLE III
THREE REAL DATASETS USED IN OUR EXPERIMENTS

Label	Dataset	Polygons	Segments	Size
Urban	*ne_10m_urban_areas*	11,878	1.1M	20MB
	ne_10m_states_provinces	4,647	1.3M	50MB
Water	*USA_Water_Bodies*	463,591	24M	520MB
	USA_Block_Boundaries	219,831	60M	1300MB
Lakes	*Lakes*	7.5M	277M	9GB
	Sports	1.8M	20M	590MB

B. Hardware Description

We have used Intel Xeon E5-2695 multi-core CPU with 45MB cache and base frequency of 2.10GHz. We have used two different kinds of GPU to run the experiments, namely, Titan V and Titan Xp. Titan V is more powerful GPU and its architecture is NVidia Volta. It has 640 Tensor Cores, 12 GB HBM2 memory, 5120 CUDA Cores and its memory bandwidth is 652.8GB/s. Architecture of Titan Xp is Pascal. It has 12 GB GDDR5X memory, 3840 CUDA Cores and its memory bandwidth is 547.7GB/s. For experiments on a single GPU, we have used Titan V. When using multi-GPUs, we used one Titan V and three Titan Xp. The PGI compiler version is 18.10.

First, we used classical filter and refine technique using a CPU-GPU system for LSI and PNP functions accelerated by OpenACC pragmas. R-tree is used for filtering on a CPU. Table IV shows the results. Even with a powerful GPU, it takes about 44 seconds in total for the larger dataset. This is the motivation behind developing a hierarchical filter and refinement system.

TABLE IV
RUNNING TIMES BY USING DIFFERENT GPUS WITHOUT FILTERS

		Water	Urban
Titan V	LSI function (s)	10.47	0.4
	PNP function (s)	33.44	0.99
	Total time (s)	43.91	1.39
Titan XP	LSI function (s)	22.16	0.82
	PNP function (s)	92.99	2.51
	Total time (s)	115.15	3.33

C. PolySketch and CMBR Results

For analysis, let us consider that each task has two polygons; P from layer 1 and Q from layer 2. For one task, P has m line segments and Q has n line segments. For LSI function, every line segment from P should be compared with all line segments from Q. The workload is $m * n$ for every task. Therefore, the total workload for LSI function is the summation of workload of individual tasks. The Application of CMBR or $PolySketch$ filter decreases the line segments in each task. This leads to workload reduction.

Tables V, VI, and VII show $PolySketch$ and CMBR's effect on reducing the workload and the number of line segments for the LSI function for different datasets. (In the tables, Sketch means $PolySketch$) We can see that both CMBR and $PolySketch$ can reduce a lot of line segments and the workload. $PolySketch$ works better than CMBR overall to reduce the total workload which is directly related to the running time. Therefore, we can get better execution time results by using $PolySketch$.

Another advantage of $PolySketch$ is that it is easier to implement using compiler directives. We only need to record overlapping tiles. Therefore, we implemented both $PolySketch$ and LSI function together using OpenACC. For implementing CMBR, we need to calculate their CMBRs, test and store the line segments that overlap with the CMBRs. We implemented CMBR to preprocess data on CPU and run LSI function on GPU.

To be fair, we show CMBR time and its LSI function time separately for CMBR Filter so that we can see how much time LSI function took. We show execution time for $PolySketch$ construction and LSI function together for $PolySketch$ Filter. For the bigger data, $PolySketch$ method's time which includes $PolySketch$ time and its LSI function time is better than CMBR method's LSI function time which does not include CMBR time. For other data sets, $PolySketch$ method's times are similar to CMBR method's LSI function times which do not include CMBR time.

Table V shows the result for Water dataset. We can see that $PolySketch$ is more effective in reducing the workload compared to CMBR. As we mentioned before, for some polygon pairs, their CMBRs can be very large. This leads to less effective filtering of line segments in those CMBRs, which in turn increases the workload in the refinement phase.

TABLE V
SKETCH AND CMBR EFFECT ON THE LSI FUNCTION FOR WATER DATASET

Water	No filters	With CMBR	With Sketch
Time(s)	10.47	10.36 + 4.53	1.39
workload	411,876,982,358	16,327,012,938	1,789,226,826
# of segments (L1)	1,036,879,194	26,844,066	242,685,263
# of segments (L2)	1,996,217,931	30,765,554	145,134,707
# of tasks	1,020,458	274,283	321,658

If we only consider the number of line segments present in each individual layer after the application of CMBR filter, we can see that CMBR is quite effective in this scenario. This is due to the fact that when the overlap area between two polygons is small, their CMBR will have fewer line segments.

When we consider line segment reduction in a single layer case, $PolySketch$ is less effective, even though it is quite effective in workload reduction compared to CMBR. This discrepancy can be explained by the way we count the number of line segments in a tile after the filter phase. For $PolySketch$, since one tile of a polygon may overlap with more than one tile of another polygon. Therefore, when counting the number of line segments in a tile after using $PolySketch$ filter, we count those line segments more than once. However, to calculate the workload, we need to consider the line segments in all the candidate pairs from both layers. Workload in LSI function directly affects the execution time. Table V also shows that the execution time of using $PolySketch$ + LSI function is even smaller than the execution time of CMBR + LSI function.

TABLE VI
SKETCH AND CMBR EFFECT ON THE LSI FUNCTION FOR URBAN DATASET

Urban	No filters	With CMBR	With Sketch
Time(s)	0.4	0.23 + 0.03	0.06
workload	6,453,160,088	25,737,640	7,489,801
# of segments (L1)	3,497,270	914,074	834,146
# of segments (L2)	65,476,891	78,492	847,581
# of tasks	28,687	8,166	9,729

TABLE VII
SKETCH AND CMBR EFFECT ON THE LSI FUNCTION FOR LAKES DATASET

Lakes	No filters	With CMBR	With Sketch
Time(s)	2.20	9.4 + 0.51	1.17
workload	29,289,344,523	260,210,378	37,464,000
# of segments (L1)	1,932,905,302	4,061,067	7,716,460
# of segments (L2)	76,801,765	1,763,838	6,143,938
# of tasks	692,435	132,888	201,107

Table VI shows the results for Urban dataset. We can see that $PolySketch$ works better than CMBR in reducing the total workload. For $PolySketch$ method, the running time which includes $PolySketch$ time and LSI time is similar to the time of LSI function after using CMBR Filter. Table VII shows the results for Lakes dataset. For Lakes dataset as well, $PolySketch$ reduces a considerable amount of workload compared to CMBR.

One of the intentions of these two filters is to discard the invalid tasks for LSI function so we can use GPU efficiently only for the valid tasks where two polygons may have line segment intersection(s). To see our filter's efficiency in discarding invalid tasks for LSI function, we define its efficiency as

$$D_{\text{Task}} = \frac{\text{The number of tasks discarded}}{\text{The original number of tasks}} \tag{1}$$

TABLE VIII
SKETCH AND CMBR EFFECT ON REDUCING TASKS OF THE LSI FUNCTION FOR DIFFERENT DATASETS

	Water	Urban	Lakes
CMBR	73.13%	71.53%	80.81%
Sketch	68.48%	66.09%	70.96%

Table VIII shows CMBR and $PolySketch$ efficiency percentage for discarding invalid tasks. We can see that both CMBR and $PolySketch$ can discard most of the tasks for LSI function and CMBR is more effective in comparison. However, we use $PolySketch$ to reduce tasks as well as workload. This is due to the fact that our compiler directive based CMBR filter implementation is slower compared to $PolySketch$ filter implementation.

For quantitative evaluation, here we describe the equations for workload and line segment reduction. In the equations below, C is the candidate set (task), for a candidate pair (i,j), E_i and E_j are the number of the line segments in ith and jth polygons. The symbols with hat notation show the reduced number of line segments due to hierarchical filtering. Using these symbols, we define the workload reduction percentage and line segment reduction percentage as

$$RP_{\text{Workload}} = \left(1 - \frac{\sum_{(i,j)\in C} \left|\widehat{E}_i\right| * \left|\widehat{E}_j\right|}{\sum_{(i,j)\in C} |E_i| * |E_j|}\right) * 100\% \tag{2}$$

and

$$RP_{\text{Line-Segment}} = \left(1 - \frac{\sum_{i\in C} \left|\widehat{E}_i\right|}{\sum_{i\in C} |E_i|}\right) * 100\% \tag{3}$$

TABLE IX
SKETCH EFFECT ON REDUCING WORKLOAD AND LINE SEGMENTS BY THE LSI FUNCTION FOR THREE DATASETS USING PERCENTAGE.

Sketch for LSI Function	The workload reduction percentage	The segments reduction percentage for L1	The segments reduction percentage for L2
Water	99.57%	76.59%	92.73%
Urban	99.88%	76.15%	98.7%
Lakes	99.87%	99.6%	92%

Tables IX and X show the effect of $PolySketch$ and CMBR Filter in reducing workload and line segments of each layer

TABLE X
CMBR EFFECT ON REDUCING WORKLOAD AND LINE SEGMENTS BY THE LSI FUNCTION FOR THREE DATASETS USING PERCENTAGE.

CMBR for LSI Function	The workload reduction percentage	The segments reduction percentage for L1	The segments reduction percentage for L2
Water	96.04%	97.41%	98.46%
Urban	99.6%	73.86%	99.88%
Lakes	99.11%	99.79%	97.7%

for the LSI function for three datasets. $PolySketch$ Filter also reduces the number of line segments from both layers. In some cases, it can discard more line segments from a layer where CMBR Filter is not so effective. In addition, $PolySketch$ Filter can also reduce more workload compared to CMBR which is more related to the execution time.

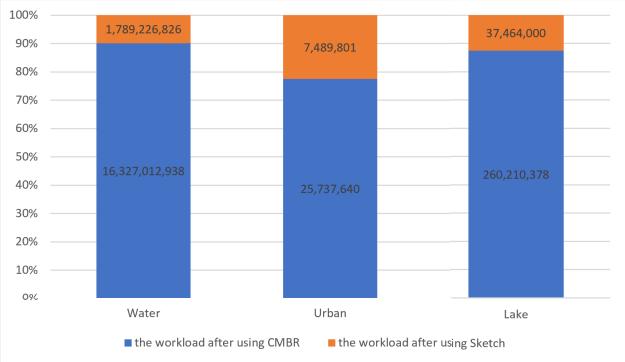


Fig. 8. The workload in LSI function after using CMBR or Sketch.

Figure 8 shows the workload for LSI function after using CMBR or $PolySketch$ Filter. As we can see that $PolySketch$ Filter works well in reducing more workload compared with CMBR Filter. For the Water dataset, we can see that the workload after using $PolySketch$ is 11% of the workload after using CMBR. For the Urban dataset, the workload after using $PolySketch$ is 29% of the workload after using CMBR. For the Lake dataset, the workload after using $PolySketch$ is 14% of the workload after using CMBR.

For the Water dataset, the number of thread blocks chosen by PGI compiler was 65535 and the number of threads in a block was 128 for $PolySketch$ with LSI function. Even when the number of tasks was greater than 65535, PGI compiler generated the grid with 65535 thread blocks.

D. Different PolySketch size Results

The real-world datasets include small as well as large polygons. Finding an ideal tile-size for a $PolySketch$ Filter is difficult. Tile-size of a sketch means the number of line segments in a tile for $PolySketch$ Filter algorithm. Table XI shows the performance of using different tile-size for the Water dataset. We can either use the same tile-size for both polygons or use different tile-size. Based on our experience, we recommend different tile-size for small and large polygons

TABLE XI
THE PERFORMANCE OF USING DIFFERENT TILE-SIZE FOR WATER DATASET

Water	Current workload	Current tasks	Time(s)
15 (5)	1,789,226,826	321,658	1.39
15 (10)	1,875,845,026	340,303	1.49
15	1,970,120,151	355,172	1.55
20 (10)	2,554,936,706	356,818	1.62
20	2,772,593,129	385,327	1.82
30	4,460,548,392	442,279	2.29

in a task. For example, in our experiments, we set tile-size as 15 for large polygons and 5 for small polygons. In our experiment, we found that if the number of line segments of a polygon is smaller than 400, tile-size of 5 worked well. For larger polygons, the tile-size of 15 worked well in reducing the workload and execution time. As shown in Table XI, in the first column, the first number is a tile-size. If there is a bracket, it means we used two tile-sizes and the number in the bracket is the tile-size used for the small polygon. The second column shows the current workload for LSI function after using $PolySketch$ Filter. The third column shows the number of valid tasks for LSI function after using $PolySketch$. The fourth column is the execution time of running $PolySketch$ and LSI function together. We can see that smaller tile-size works better. It can reduce more workload and discard more invalid tasks. The execution time is also less. In general, if the tile-size is small enough, we can discard more tile overlap pairs and only use LSI function for the overlapping tile pairs. Finally, for Water and Lakes, we set tile-size as 15 for large polygons and 5 for small polygons. For Urban dataset, we set tile-size as 10.

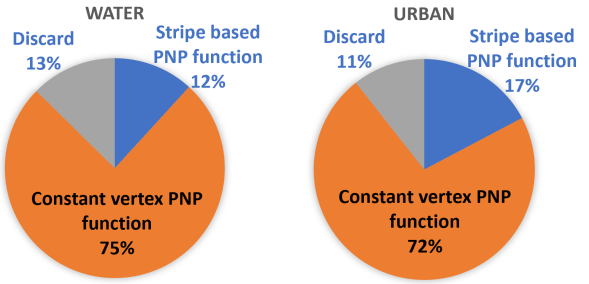
E. PNP Based Task Reduction Algorithm and Striping Algorithm Results

TABLE XII
PNP BASED TASK REDUCTION ALGORITHM AND STRIPING EFFECT ON REDUCING WORKLOAD OF THE PNP FUNCTION FOR BOTH THE DATASETS AND THE REDUCTION PERCENTAGE

	Original workload	Current workload	Reduction percentage
Water	411,876,982,358	15,653,774,431	96.2%
Urban	6,453,160,088	291,816,678	95.48%

Table XII shows PNP based task reduction algorithm and striping effect on reducing workload of the PNP function for both the datasets and the reduction percentage. We can see that it reduced most of the workload. One reason is that we can discard some tasks where two polygons do not have any line segment intersection and the bigger MBR does not contain the smaller MBR. Otherwise, there are only two types of tasks where we need to use PNP function. One case is two polygons have line segment intersection(s) so there should be some vertices which are inside another polygon. Another case is when two polygons do not have any line segment intersection but the bigger MBR contains the smaller MBR so one polygon may be totally inside another polygon.

Although we need to use PNP function for these tasks, we have appropriate filters and refinement steps for them. For the first case, we use Striping method to reduce the vertices, line segments and workload. Striping can be effective when the CMBR of two polygons is large. For the second case, we check only a few vertices of two polygons to see which polygon is inside or outside another polygon based on our analysis. We only need to check a few vertices of only one polygon for PNP function because the bigger polygon cannot be inside the smaller polygon.



(a) (b)

Fig. 9. The percentage of different types of tasks after using PNP based task reduction algorithm

Figure 9 shows the percentage of different types of tasks after using PNP based task reduction algorithm. We can see that PNP based task reduction algorithm is very efficient. For Water dataset, we need to check all vertices of polygons only for 12% of the tasks because the polygons of these tasks have line segment intersection(s). However, we can use Striping method which is very helpful to reduce the workload of these tasks and discard some vertices which cannot be inside another polygon. Then, we use Stripe-based PNP function. In addition, for 75% of the tasks, we can only use constant vertex PNP function (5 vertices) of a polygon to determine whether they are inside or outside another polygon and then we know whether this polygon is inside or outside of another polygon according to our analysis. We can also discard 13% of the tasks and do not need to do PNP tests for these tasks. PNP based task reduction algorithm also works well for Urban dataset. We can discard 11% of the tasks and perform constant vertex PNP function for 72% of the tasks. Then, we use Stripe-based PNP function for the remaining 17% of the tasks.

TABLE XIII
STRIPING EFFECT ON REDUCING WORKLOAD FOR THE STRIPE-BASED PNP FUNCTION FOR BOTH THE DATASETS

	Original workload	Current workload	Reduction percentage
Water	156,443,271,335	5,301,138,126	96.61%
Urban	1,254,513,546	14,321,168	98.86%

Table XIII shows Striping algorithm effect on reducing the workload of the Stripe-based PNP function for both the datasets. Stripe-based PNP function is applied to the tasks where two polygons have line segment intersection(s). We can see that it can reduce most of the workload for both the datasets. The vertices inside a stripe need to be tested against the line segments only within the same stripe and crossing the stripe boundary, instead of all the line segments of a polygon. This leads to workload reduction. Even the area where we want to do striping is very large, we can still get benefit.

TABLE XIV
PNP BASED TASK REDUCTION ALGORITHM EFFECT ON REDUCING WORKLOAD WHEN A POLYGON MBR IS INSIDE ANOTHER POLYGON MBR

	Original workload	Current workload	Reduction percentage
Water	152,287,577,854	10,352,636,305	93.2%
Urban	4,788,726,632	277,495,510	94.2%

In case when a polygon is inside another polygon, the MBR of small polygon is inside the MBR of larger polygon. Our PNP-based task reduction algorithm detects these cases. So, we do not apply the quadratic time PNP tests in these cases. This results in workload reduction compared to the naive cases. Table XIV shows PNP based task reduction algorithm's effect on reducing workload of these cases. We only check five vertices of the smaller polygon to see whether these vertices are inside or outside of another polygon. Then, we determine whether the smaller polygon is inside or outside of the bigger polygon.

F. Execution time details

TABLE XV
EXECUTION TIME IN SECONDS

Dataset	1 CPU thread and 1 GPU	32 CPU threads, 1 GPU for LSI and PNP	32 CPU threads, 1 GPU for LSI, multi-GPUs for PNP
Urban	1.39	0.35	0.30
Water	43.92	10.63	7.71

Table XV shows the total running times for two datasets. The first column's result is using 1 thread on CPU and 1 GPU without using our hierarchical filtering. For testing our system, we used one or more threads on CPU and one or multiple GPUs to see the difference. The second column's result is using 32 threads on CPU to preprocess data and 1 GPU for LSI and PNP function. The third column's result is using 32 threads on CPU to preprocess data, 1 GPU for LSI function and multi-GPUs for PNP function. We can see that our system works well using multi-core CPU and multiple GPUs. Although we only use multi-GPUs for PNP function, we can still get benefit, especially for the larger dataset.

Tables XVI and XVII show the details of execution time breakdown for the two datasets. We can see that our filters are very efficient and we can get benefit by using multi-GPUs. If the dataset is larger, we can get more benefit by using multi-GPUs. For Water dataset, the time taken by R-tree filter on CPU is 2.27s. Therefore, the end-to-end time of the system is 9.98s.

TABLE XVI
EXECUTION TIME BREAKDOWN DETAILS FOR WATER DATASET. (NA MEANS IT IS NOT APPLICABLE.)

Water	Sketch and LSI Function on GPU (s)	Pre-process for PNP on CPU(s)	PNP Function on GPU (s)	Final Time (s)
No filters	8.82	NA	35.1	43.92
One GPU for LSI and PNP	1.39	4.56	4.68	10.63
One GPU for LSI, multi-GPUs for PNP	1.39	4.55	1.77	7.71

TABLE XVII
EXECUTION TIME BREAKDOWN DETAILS FOR URBAN DATASET. (NA MEANS IT IS NOT APPLICABLE.)

Urban	Sketch and LSI Function on GPU (s)	Pre-process for PNP on CPU(s)	PNP Function on GPU (s)	Final Time (s)
no filters	0.4	NA	0.99	1.39
One GPU for LSI and PNP	0.08	0.19	0.08	0.35
One GPU for LSI, multi-GPUs for PNP	0.07	0.19	0.04	0.30

VI. CONCLUSION AND FUTURE WORK

We have developed a hierarchical PolySketch-based filter and refine system for GPUs and evaluated its performance using real-world datasets. Even though the system was implemented using compiler directives, the performance is very good. Spatial join on two large datasets can be performed in about 10 seconds. This is an order of magnitude better performance than our previous work where we did not leverage hierarchical filtering [4].

We plan to integrate our GPU-accelerated system to MPI-GIS and MapReduce implementations which we have built as an HPC system for geospatial analytics [11], [20]–[23]. PNP algorithm can be further improved by doing the pre-processing on GPU. We also plan to improve $PolySketch$ by making the tile-size adaptive.

VII. ACKNOWLEDGEMENT

This work is partly supported by the National Science Foundation CRII Grant No.1756000. We gratefully acknowledge the support of NVidia Corporation with the donation of the Titan X Pascal GPU used for this research.

REFERENCES

[1] A. Aji, G. Teodoro, and F. Wang, "Haggis: turbocharge a MapReduce based spatial data warehousing system with GPU engine," in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. ACM, 2014, pp. 15–20.

[2] J. Nievergelt and F. P. Preparata, "Plane-sweep algorithms for intersecting geometric figures," *Communications of the ACM*, vol. 25, no. 10, pp. 739–747, 1982.

[3] M. McKenney and T. McGuire, "A parallel plane sweep algorithm for multi-core systems." in *GIS*, 2009, pp. 392–395.

[4] A. Paudel and S. Puri, "OpenACC Based GPU Parallelization of Plane Sweep Algorithm for Geometric Intersection," in *International Workshop on Accelerator Programming Using Directives*. Springer, 2018, pp. 114–135.

[5] A. Margalit and G. D. Knott, "An algorithm for computing the union, intersection or difference of two polygons," *Computers & Graphics*, vol. 13, no. 2, pp. 167–183, 1989.

[6] J. Zhang and S. You, "Speeding up large-scale point-in-polygon test based spatial join on GPUs," in *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. ACM, 2012, pp. 23–32.

[7] S. K. Prasad, S. Shekhar, M. McDermott, X. Zhou, M. Evans, and S. Puri, "GPGPU-accelerated interesting interval discovery and other computations on geospatial datasets: A summary of results," in *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. ACM, 2013, pp. 65–72.

[8] S. K. Prasad, D. Aghajarian, M. McDermott, D. Shah, M. Mokbel, S. Puri, S. J. Rey, S. Shekhar, Y. Xe, R. R. Vatsavai *et al.*, "Parallel processing over spatial-temporal datasets from geo, bio, climate and social science communities: A research roadmap," in *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 2017, pp. 232–250.

[9] S. Audet, C. Albertsson, M. Murase, and A. Asahara, "Robust and efficient polygon overlay on parallel stream processors," in *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2013, pp. 304–313.

[10] S. Puri and S. K. Prasad, "Output-sensitive parallel algorithm for polygon clipping," in *2014 43rd International Conference on Parallel Processing*. IEEE, 2014, pp. 241–250.

[11] ——, "A parallel algorithm for clipping polygons with improved bounds and a distributed overlay processing system using mpi," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 576–585.

[12] D. Aghajarian, S. Puri, and S. Prasad, "GCMF: an efficient end-to-end spatial join system over large polygonal datasets on GPGPU platform," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2016, p. 18.

[13] D. Aghajarian and S. K. Prasad, "A spatial join algorithm based on a non-uniform grid technique over GPGPU," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2017, p. 56.

[14] C. Gao, F. Baig, H. Vo, Y. Zhu, and F. Wang, "Accelerating Cross-Matching Operation of Geospatial Datasets using a CPU-GPU Hybrid Platform," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3402–3411.

[15] E. H. Jacox and H. Samet, "Spatial join techniques," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 1, p. 7, 2007.

[16] S. K. Prasad, M. McDermott, X. He, and S. Puri, "GPU-based Parallel R-tree Construction and Querying," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 2015, pp. 618–627.

[17] J. A. Orenstein, "Redundancy in spatial databases," in *ACM SIGMOD Record*, vol. 18, no. 2. ACM, 1989, pp. 295–305.

[18] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: the international journal for geographic information and geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[19] P. Schneider and D. H. Eberly, *Geometric tools for computer graphics*. Elsevier, 2002.

[20] D. Agarwal, S. Puri, X. He, and S. K. Prasad, "A system for GIS polygonal overlay computation on linux cluster-an experience and performance report," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 1433–1439.

[21] S. Puri, A. Paudel, and S. K. Prasad, "MPI-Vector-IO: Parallel I/O and partitioning for geospatial vector data," in *Proceedings of the 47th International Conference on Parallel Processing, ICPP*, 2018, p. 13.

[22] S. Puri, D. Agarwal, X. He, and S. K. Prasad, "MapReduce algorithms for GIS polygonal overlay processing," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 2013, pp. 1009–1016.

[23] S. Puri, D. Agarwal, and S. K. Prasad, "Polygonal overlay computation on Cloud, Hadoop, and MPI," *Encyclopedia of GIS*, pp. 1–9, 2015.