The VLDB Journal (2019) 28:73–98
https://doi.org/10.1007/s00778-018-0519-4

REGULAR PAPER

# A scalable spatial skyline evaluation system utilizing parallel independent region groups

Wenlu Wang[1] · Ji Zhang[1] · Min-Te Sun[2] · Wei-Shinn Ku[1]

Received: 3 October 2017 / Revised: 8 August 2018 / Accepted: 4 September 2018 / Published online: 17 September 2018


**Abstract**
This research presents two parallel solutions to efficiently address spatial skyline queries. First, we propose a novel concept called independent regions for parallelizing the process of spatial skyline evaluation. Spatial skyline candidates in an independent region do not depend on any data point in other independent regions. Then, we propose a GPU-based solution. We use multi-level independent region group-based parallel filter to support efficient multi-threading spatial skyline non-candidate elimination. Beyond that, we propose comparable region to accelerate non-candidate elimination in each independent region. Secondly, we propose a MapReduce-based solution. We generate the convex hull of query points in the first MapReduce phase. In the second phase, we calculate independent regions based on the input data points and the convex hull of the query points. With the independent regions, spatial skylines are evaluated in parallel in the third phase, in which data points are partitioned by their associated independent regions in map functions, and spatial skyline candidates are calculated by reduce functions. The results of the spatial skyline queries are the union of outputs from the reduce functions. Our experimental results show that GPU multi-threading scheme is very efficient on small-scale input datasets. On the contrary, MapReduce scheme performs very well on large-scale input datasets.

**Keywords** Spatial skyline query · MapReduce · Parallel computation · GPU

## 1 Introduction

Since the skyline operator was introduced into database research [1], a number of efficient algorithms have been proposed for the skyline evaluation. Bitmap [2], Index [2], NN (Nearest Neighbor) [3] and BBS (Branch-and-Bound Skyline) [4] rely on indices constructed before query processing, while BNL (Block Nested Loop) [1], D&C (Divide and Conquer) [1], SFS (Sort Filter Skyline) [5] and OSPS (Object-based space partitioning skyline) [6] use non-index techniques. Moreover, several studies primarily focus on the skyline query in a variety of problem settings (data residing in a data stream [7] or on mobile devices [8]).

As a novel type of skyline query, spatial skyline query (SSQ) was proposed to consider the preference of both static and dynamic object attributes in multi-criteria decision-making applications [9]. Unlike skyline queries that only take static object attributes (e.g., rating and price of restaurants) into account, the distance between objects is also calculated as dynamic attributes in the spatial skyline queries. In particular, given a set of data points $P$ and a set of query points $Q$ in a $d$-dimensional space, spatial skyline queries return a subset of $P$, in which data points are not spatially dominated by other data points in $P$. The spatial dominance is defined by using the distance from data points to all query points.

Spatial skyline query is applicable to many applications. Taking crisis management applications as an example, we assume that a number of waterborne infectious disease cases were confirmed at different locations; people who live at spatial skyline places with respect to those locations should be

✉ Wei-Shinn Ku
weishinn@auburn.edu

Wenlu Wang
wenluwang@auburn.edu

Ji Zhang
jizhang@auburn.edu

Min-Te Sun
msun@csie.ncu.edu.tw

[1] Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA

[2] Department of Computer Science and Information Engineering, National Central University, Taoyuan County, Taiwan, ROC

alerted and examined first, because there might be a higher possibility that these people may have been exposed to contaminated water. Another example is the travel planning applications. People may prefer spatial skyline hotels with respect to fixed locations of beaches and theaters for vacation. In this case, people would prefer not to choose a hotel far from all interesting attractions. One more example of spatial skyline query is that people may plan to have dinner with their friends. They may consider the distance from their homes to various restaurants for restaurant selection. The restaurants far from all of their homes would not be included in the candidate list, because most people want to save time on the road. Thus, having a list of spatial skyline restaurants is the first step toward restaurant selection.

Two index-based algorithms were proposed to efficiently address the spatial skyline queries [9]. Branch-and-Bound Spatial Skyline $B^2S^2$ algorithm searches spatial skyline candidates by visiting an R-tree from top to bottom. Once a spatial skyline is found, $B^2S^2$ expands the R-tree to access the node which has minimum *mindist* value, and compares it with all spatial skyline candidates found up to this point in spatial dominance test. The other method, Voronoi-based Spatial Skyline $VS^2$ algorithm, relies on a Voronoi diagram created over input data points. $VS^2$ starts with the closest data points to query points and then searches in the space by visiting the neighbors of visited data points over the Voronoi diagram. Due to high cost of the spatial dominance test, $VS^2$ was improved by reducing the number of spatial dominance tests in [10]. In the method, seed skyline points (a subset of spatial skyline points) can be identified with spatial dominance test.

However, the following problems motivate us to apply parallel solutions to spatial skyline evaluation. First, the distance between moving objects may keep changing. If indices are created at a preprocessing stage, the cost of index maintenance would be unacceptably high. Secondly, as data grow rapidly, addressing skyline queries on large-scale datasets in a single-node environment becomes impractical. There are increasing numbers of approaches proposed for processing skyline queries in distributed and/or parallel environments [11]. Although distributed spatial skylines were well studied, none of the parallel spatial skyline solutions aiming to support large-scale input were reported so far. Distributed spatial skyline (DSS) is able to support spatial skyline in wireless sensor networks containing hundreds of nodes, but large-scale input data cannot be handled in that case. Thirdly, multiple parallel schemes have been applied to general skyline computation. GPU has been utilized for skyline computation in [6,12–14]. MapReduce framework has been incorporated into parallel solutions for skyline computation [15–17] and other database applications [18–20].

Therefore, we propose a scalable system consisting of two parallel solutions based on GPU multi-threading and MapReduce schemes. To parallelize the spatial skyline computation, we propose a novel concept, independent regions, in each of which spatial skylines do not depend on any data point outside the independent region. If data points do not fall in any independent region, they can be discarded because they must be spatially dominated by any data point in independent regions. We also introduce independent region group (IRG) concept, which is the union of the independent regions produced by the same data points.

In our GPU-based solution, we propose a multi-level IRG-based parallel filter, which will eliminate the majority of non-candidate data points in parallel. For example, a data point will be eliminated if it does not fall into any independent region. Each independent region is processed in parallel. After the filtering process, we adopt a comparable region concept to further avoid dominance tests on non-comparable data point pairs. Experiments show that GPU-based spatial skyline evaluation is very efficient. However, a GPU-based solution has to move data back and forth between CPU and GPU memories. As a result, the size of input data is limited by hardware capacity.

For a large dataset that cannot fit into a GPU or a CPU memory, we propose a novel three-phase MapReduce-based solution. In particular, given a set of data points and a set of query points, we calculate the convex hull of the query points in the first phase. Then, our solution produces the independent regions at the second phase. With the independent regions in the third phase, map functions associate data points with their independent regions, and reduce functions find the spatial skylines in independent regions in parallel. We also propose a novel concept, pruning regions, in independent regions. The pruning regions are the areas in which all data points are dominated by other data points. If a data point is in a pruning region, the data point can be discarded immediately.

In short, the contributions of this study are summarized below:

1. We introduce a concept of independent regions in our solution. The spatial skyline candidates in an independent region do not depend on any data points in other independent regions. With the feature of independence, spatial skyline queries can be addressed in parallel.
2. We propose a GPU-based parallel solution to efficiently evaluate spatial skyline queries on small-scale datasets.
3. We propose a MapReduce-based parallel solution to efficiently evaluate spatial skyline queries on large-scale datasets.
4. We evaluate the performance of the proposed solutions through extensive experiments with different cardinality of datasets.

The rest of this paper is organized as follows. In Sect. 2, we define the spatial skyline queries and relevant techniques

utilized in our solutions. In Sect. 3, the unique properties we discovered are formally presented. In Sect. 4, we present our advanced GPU-based solution. In Sect. 5, our advanced MapReduce-based solution is presented. The experimental validation of our design is presented in Sect. 6. Section 7 surveys related works. We conclude the paper in Sect. 8.

## 2 Preliminary

In this section, we present properties of spatial skyline queries, which can be used for pruning the search space of the query and parallelizing the query evaluation. Then, we describe key characteristics of GPUs.

### 2.1 Problem statement

Given a dataset $P$ in a $d$-dimensional space $\mathbb{R}^d$, an object $p \in P$ can be represented as $p = \{x_1, x_2, ..., x_d\}$ where $p.x_i$ is the value of the object on the $i^{th}$ dimension. $D(.,.)$ denotes a distance metric that obeys the triangle inequality in $\mathbb{R}^d$. The spatial dominance relationship and the spatial skyline operator are defined as follows [9]. All notations used in this paper are summarized in Table 1.

**Definition 1** (*Spatial Dominance*) Given a set of query points $Q$, and two data points $p$ and $p'$ in $\mathbb{R}^d$, $p$ spatially dominates $p'$ with respect to $Q$, denoted by $p \prec^Q p'$, if $\forall\, q \in Q,\ D(p,q) \leq D(p',q)$ and $\exists\, q' \in Q,\ D(p,q') < D(p',q')$.

**Table 1** Symbolic notations

| Symbol | Meaning |
|---|---|
| $P$, $Q$ | A set of data points and a set of query points |
| $p$, $q$ | A data point and a query point |
| $pi$ | Pivot point |
| $p.x_i$ | The value of data point $p$ in the $i^{th}$ dimension |
| $\mathbb{R}^d$ | A $d$-dimensional space |
| $h$ | A hyper-place |
| $S$ | A half-space |
| $F$ | A facet of a convex hull |
| $A_q^{\triangle}$ | A set of adjacent convex points of $q$ |
| $p \prec^Q p'$ | $p$ spatially dominates $p'$ with respect to $Q$ |
| $SSKY(P, Q)$ | Spatial skylines of $P$ with respect to $Q$ |
| $CH(Q)$ | The convex hull of $Q$ |
| $DR(p, Q)$ | The dominator region of $p$ with respect to $Q$ |
| $PR(p, q)$ | The pruning region generated by $p$ and $q$ |
| $IR(p, q)$ | The independent region generated by $p$ and $q$ |
| $IRP$ | Independent Regions Pivot, the independent regions are generated by $IRP$ |
| $lssky$ | A set of local spatial skyline candidates |
| $chsky$ | A set of spatial skyline candidates in a convex hull |
| $L^l\ IRG$ | An independent region group with depth $l$ |

**Definition 2** (*Spatial Skyline*) Spatial skylines of a set of data points $P$ with respect to a set of query points $Q$ in $\mathbb{R}^d$, denoted by $SSKY(P, Q)$, are a set of data points in $P$, which are not spatially dominated by any other data point in $P$ with respect to $Q$.

$$SSKY(P, Q) = \{p \in P \mid \nexists\, p' \in P, p \neq p', p' \prec^Q p\} \tag{1}$$

**Property 1** *If any data point $p \in P$ is a spatial skyline point with respect to a subset of query points $Q' \subset Q$, then $p$ is also a spatial skyline point with respect to $Q$* [9].

**Property 2** *The set of spatial skyline points of data points $P$ does not depend on any non-convex query points $q \in Q$, $q \notin CH(Q)$, where $CH(Q)$ indicates the convex hull of $Q$* [9]. *In other words,*

$$SSKY(P, Q) = SSKY(P, CH(Q)) \tag{2}$$

**Definition 3** (*Dominator Region*) Given a data point $p \in P$, a set of query points $Q$ and hyper-spheres that center at $q_i$ with radius $D(p, q_i)$, $q_i \in Q$, any data point inside the intersection of the hyper-spheres spatially dominates $p$ with respect to $Q$. The intersection area that potentially contains data points spatially dominating $p$ with respect to $Q$ is referred to as the dominator region of $p$, denoted by $DR(p, Q)$.

Dominator region enables our solution to efficiently eliminate data points by reducing the search space of data points. For example, Fig. 1 displays the dominator region of a data point $p$ and a set of query points $Q$. $Q$ has three query points $q_1$, $q_2$ and $q_3$, which represent a convex hull in a 2-dimensional space. Three circles centered at $q_i \in Q$ with radius $D(q_i, p)$ are created in order to highlight the dominance areas of $p$ with respect to the query points. Any data point $p'$ in the intersection of the three circles spatially dominates $p$ with respect to $Q$.

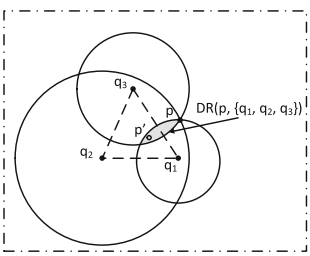


**Fig. 1** An example of $DR(p, \{q_1, q_2, q_3\})$ in a 2-dimensional space

### 2.2 GPU computation overview

A graphics processing unit (GPU) is able to provide tremendous opportunities for computational parallelism using thousands of multi-threaded processing cores. The features of GPU architecture are summarized as follows:

*Computational model.* Benefiting from a combination of multi-threaded physical cores with rapid context switching, a GPU is able to reach teraflops level computational throughput. GPUs follow Single Instruction Multiple Thread (SIMT) computing paradigm, and multiple threads are grouped together, called a warp. Then, the threads in the warps are mapped to Single Instruction Multiple Data (SIMD) execution unit—i.e., all threads within the same wrap execute the same instruction, but with different data. At the last step, similar to MapReduce scheme, GPU computation has a reduce phase, which performs a summary operation over all the results from every thread. In general, there are two types of GPU standard reduction: serial reduction and parallel reduction. Most of the works, including ours, use parallel reduction to increase efficiency.

Within a warp, all threads are step-locked. *Branch divergence* is a phenomenon that multiple threads within the same warp go to different branches due to branch instructions. Such divergence serializes computation because some threads are idle, while the processing core is executing for the other branch.

*Latency hiding.* The combination of fast context- switching and the large number of warps is able to hide memory accessing latencies, which is called *latency hiding*. Whenever the execution of a warp is stalled, another warp can jump in and get executed immediately to maximize the overall computing resources utilization.

*Memory model.* A GPU suffers from high latencies when copying data between CPUs (host) and GPUs (device). The GPU memory hierarchy consists of a global memory shared among all resources, an L2 cache, and several lower-level caches. Each streaming multiprocessor has 64KB configurable shared memory and L1 cache. The latency of shared memory is approximately 100 times lower than the one of global memory. However, it is a small-size memory that only local to a thread block.

### 2.3 MapReduce overview

MapReduce was proposed as a generic programming model for data-intensive applications in distributed environments [21]. The framework provides two simple primitives, *map* and *reduce* functions, and allows developers to focus mainly on their functionality. The task scheduling, load balancing and other issues are encapsulated in the MapReduce framework, which significantly reduces the difficulty of the development of parallel applications. Driven by the MapReduce framework, *map* functions receive data in key/value pairs from input streams and output intermediate results in another type of key/value pairs. Then, *reduce* functions retrieve the intermediate results and write final results to an output stream. In the shuffle phase, the intermediate results are automatically grouped and sorted by the MapReduce framework, The two primitives can be represented as: map $(K_1, V_1) \rightarrow \text{list}(K_2, V_2)$ and reduce$(K_2, \text{list}(V_2)) \rightarrow \text{list}(K_3, V_3)$.

### 2.4 Convex hull and spatial skyline queries

Given a set of query points $Q$ in a $d$-dimensional space $\mathbb{R}^d$, the convex hull of $Q$, denoted by $CH(Q)$, is the smallest convex polytope that contains all query points in $Q$. Theoretically, a convex hull can be represented as either a set of convex points or the intersection of a set of half-spaces. Each half-space contains all the query points in $Q$. Moreover, a convex hull can also be abstracted by a set of facets and their adjacency relationships. Each facet can be defined by a number of convex points. For example, a facet (line) can be determined by two adjacent convex points in a 2-dimensional space. The facets become planes that can be represented by a convex point and its two adjacent convex points in a 3-dimensional space. Because the facets of $CH(Q)$ separate the query points in $Q$ from any point outside the convex hull, connecting a data point $v$ outside $CH(Q)$ with any data point in $CH(Q)$ must intersect with at least one facet of the convex hull. Thus, the facet is referred to as a visible facet from $v$.

The properties of convex hull provide opportunities to optimize the process of spatial skyline evaluation by reducing the search space of both data points and query points. Given a set of data points $P$ and the convex hull of a set of query points $Q$, all data points inside $CH(Q)$ are spatial skylines of $P$ with respect to $Q$ [9]. Given two data points, if they are in the convex hull, the bisector hyper-plane of these two points partitions the space into two half-spaces, and there must exist convex points in either half-space. Thus, neither of the two data points can spatially dominate the other, and both of them are spatial skylines. If one point $p_1$ is in the convex hull and the other $p_2$ is not, then, the bisector line of $p_1$ and $p_2$ partitions the space into two half-spaces, and there must exist a convex point in the same half-space with $p_1$. If the convex point does not exist, the convex hull cannot contain $p_1$, which contradicts with our assumption. Thus, $p_1$ is not spatially dominated by $p_2$. These two cases are summarized in Property 3.

**Property 3** *Given a set of data points $P$ and a set of query points $Q$, if any point $p \in P$ is inside the convex hull of*

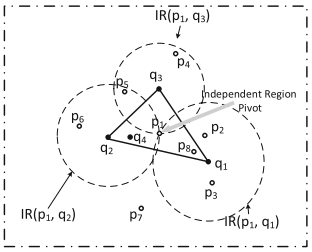


**Fig. 2** An example of independent regions in a 2-dimensional space

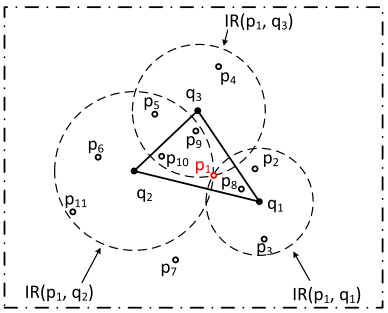


**Fig. 3** $L^1$ independent region filter.

*$Q$, then $p$ is a spatial skyline of $P$ with respect to $Q$ ($p \in SSKY(P, Q)$).*

## 3 Independent region and independent region group

In this section, we first provide a formal definition of an independent region and an independent region group. Then we explain how to select Independent Region Pivot.

**Definition 4** (*Independent Region*) Given a data point $p$ and a set of query points $Q$ in a $d$-dimensional space, we define an *Independent Region (IR)* of $p$ and $q_i$, $q_i \in Q$ as a sphere centered at $q_i$ with radius $D(p, q_i)$. An *independent region group (IRG)* of $p$ with respect to $Q$ is the union of the independent regions, as shown in Fig. 2.

$$IRG(p, Q) = \bigcup_{q_i \in Q} IR(p, q_i), \quad \text{where}$$
$$IR(p, q_i) = \{l |\ D(l, q_i) \leq D(p, q_i)\} \tag{3}$$

We define data point $p$ as the *Independent Region Pivot* of $IRG(p, Q)$ as shown in Fig. 2.

With the definition of the independent region, we provide the *independence* of spatial skylines as follows.

**Theorem 1** *Given a data point $p$ and its independent regions $\{IR(p, q_j) \mid q_j \in CH(Q)\}$, where $CH(Q)$ is the convex hull of query points $Q$, $\forall\ q_j \in CH(Q)$, any data point $p' \in IR(p, q_j)$ is not dominated by any data point $p'' \notin IR(p, q_j)$.*

***Proof*** The proof is by contradiction. Assume that $\exists\ p' \in IR(p, q_j)$, $p'' \notin IR(p, q_j)$, $p'' \prec^Q p'$. By the definition of spatial skyline, $p''$ is spatially closer to any query point $q_i$ ($q_i \in Q$) than $p$. Since $q_j \in CH(Q)$, so $q_j \in Q$ as well. But according to the definition of independent regions, $D(p'', q_j) \geq D(p', q_j)$ since $p''$ is outside of $IR(p, q_j)$, which leads to a contradiction. Thus, this concludes the proof. □

*Independent region group.* The concept of independent region group is developed for parallelizing the spatial dominance test. The dominance of objects in an independent region do not rely on any objects outside the independent region. Given a set of data points $P$ and a set of query points $Q$ in a $d$-dimensional space, an independent region group can be generated by selecting a pivot $p$ from $P$.

An independent region group specifies a smaller search space, which contains all spatial skyline candidates. The objects outside the independent region group can be eliminated without spatial dominance test because all these objects are dominated by the pivot point.

Based on Theorem 1, we find more properties regarding independent region group.

**Property 4** *Given a set of data points $P$ and a set of query points $Q$, there does not exist a point $p \in P$, where $p \in SSKY(P, Q)$ and $p \notin IRG(p, Q)$. In other words, the objects in $IRG(p, Q)$ are a superset of $SSKY(P, Q)$.*

Moreover, the independent region group also partitions the limited search space into independent regions; the dominance test in independent regions can be processed in parallel. There is no data exchange among independent regions in the process of spatial query evaluation.

**Property 5** *Given a set of data points $P$ and a set of query points $Q$, let $p$ be a pivot point, then*

$$SSKY(P, Q) = \bigcup_{IR \in IRG(p, Q)} SSKY(P_{IR}, Q) \tag{4}$$

where $P_{IR}$ represents a set of data points in the independent region $IR$.

## 4 Spatial skyline evaluation using GPU multi-threading scheme

In this section, we will introduce spatial skyline using GPU multi-threading scheme. In Sect. 4.1, we introduce a new concept of multi-level independent region group, which can be utilized as a filter step. An overview of the GPU solution is introduced in Sect. 4.2.

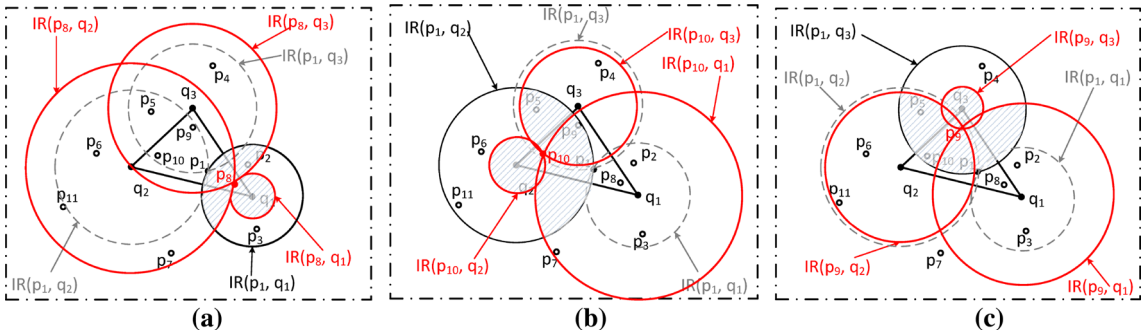


**Fig. 4** $L^2$IRG filters with varied pivots. **a** Using $p_8$ as pivot while expanding $L^1$IRG. **b** Using $p_{10}$ as pivot while expanding $L^1$IRG. **c** Using $p_9$ as pivot while expanding $L^1$IRG

### 4.1 Multi-level independent region group (MIRG)

As derived from independent region groups, multi-level independent region groups can be defined recursively as follows.

**Definition 5** ($L^1$ *independent region group*) As shown in Fig. 3, given a data point $p$ and a set of query points $Q$, the independent region group of $p$ with respect to $Q$ is defined as an $L^1$ *independent region group*, which is defined in Eq. 3. In fact, $L^1IRG(p, Q)$ is a basic building block of MIRGs.

**Definition 6** ($L^n$ *independent region group*) Given an $L^{n-1}$ independent region group, denoted by $L^{n-1}$IRG, for any $L^{n-1}$ independent region in $L^{n-1}$IRG, if there exists a data point $p'$ in the independent region, an $L^n$ *independent region group* can be generated by further overlapping every independent region with $IRG(p', Q)$. There is a special case. Suppose $L^{n-1}IR_j \in L^{n-1}IRG$, and $p' \notin L^{n-1}IR_j$, it is possible that $\exists IR(p', q')$, $L^{n-1}IR_j \in IR(p', q')$ $(IR(p', q') \in IRG(p', Q))$. In this case, $L^{n-1}IR_j$ cannot be further partitioned. For $L^{n-1}IR$s like $L^{n-1}IR_j$, they will be included into $L^nIRG$ directly without further partitioning.

$$
\begin{aligned}
&L^nIRG \\
&= \Big\{ L^{n-1}IR_j \cap IR(p', q') \,|L^{n-1}IR_j \in L^{n-1}IRG, \\
&\quad IR(p', q') \in IRG(p', Q), \nexists IR(p', q'), \\
&\quad L^{n-1}IR_j \in IR(p', q') \Big\} \bigcup \\
&\quad \Big\{ L^{n-1}IR_j \,|\exists IR(p', q'),\ L^{n-1}IR_j \in IR(p', q') \Big\}
\end{aligned}
\tag{5}
$$

Figure 4 shows an example of $L^2$ IRG generated from $L^1$ IRG in Fig. 3.

It is worth noting the following key points. (1) $L^nIRG$ is defined by a recursion. During each recursion, query set $Q$ will be iterated, and one more pivot will be selected. (2) Given an $L^{n-1}IRG$, $L^nIRG$ varies when a different pivot data point $p'$ is selected because $IRG(p', G)$ overlapped with $L^{n-1}IRG$ is different. (3) The selected pivot points may vary among $L^{n-1}$ independent regions. If a data point is in two $L^{n-1}$ independent regions, the data point could be selected as the pivot data point for both independent regions. (4) Once a new pivot is selected, the overlapping independent regions generated by the pivot point cannot be empty because the pivot is in the independent region. (5) If an $L^{n-1}$ independent region does not contain any data points, it will not be further partitioned, and $L^nIRG$ will not contain that independent region as well.

We introduce the following properties of multi-level IRG:

**Property 6** (Search space) *Given a set of data points $P$ and a set of query points $Q$, any multi-level IRG generated from $P$ and $Q$ specifies the search space of spatial skyline query of $P$ with respect to $Q$. Any object outside the IRG can be discarded without the spatial dominance test.*

**Property 7** *Given an $L^n$IRG generated from an $L^{n-1}$IRG, the search space of $L^n$IRG is equal to or smaller than that of $L^{n-1}$IRG.*

**Property 8** *(*Independence*) Given a multi-level IRG that consists of a set of independent regions. Any objects in an independent region does not rely on any objects outside the independent region.*

**Property 9** *Given an $L^n$ IRG generated from a set of data points $P$ and a set of query points $Q$, the MIRGs contain at most $(|Q| - 1) * n + 1$ independent regions.*

### 4.2 Framework of the GPU-based solution

Spatial skyline queries are evaluated in three steps in our proposed solution, displayed in Fig. 5, and the outline of our algorithm is described in Algorithm 1. Specifically, our solution receives a set of data points $P$ and a set of query points $Q$ and outputs spatial skyline points of $P$ with respect to $Q$.

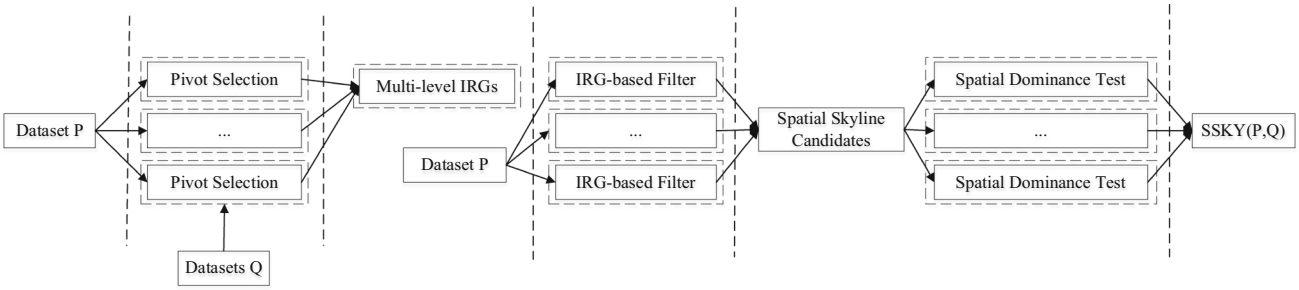


**Fig. 5** An overview of parallel spatial skyline processing using GPU

**Algorithm 1** Spatial Skyline GPU Algorithm

**Input:** $P$, $Q$
**Output:** $ssky$
1: $IRGs$ = **CalculatingPivots&IR** ($P$, $Q$);
2: $P'$ = **ParallelFilter**($P$, $IRGs$);
3: $ssky$ = **SpatialDominanceTest**($P'$, $Q$);
4: **return** $ssky$;

By utilizing the properties of independent region groups, we propose multi-level independent region groups to form a space partition tree. Each level of tree nodes represents a group of independent regions, the union of which covers the whole search space.

As we described in Sect. 3, each IRG is generated by an independent region pivot. A multi-level IRG is built on multiple IRGs. For example, $L^m$ IRG needs $m$ independent region pivots. We first generate all independent region pivots and multi-level IRG in parallel. In the parallel filter step, all data points are filtered in parallel based on independent regions in multi-level IRGs. If data point $p$ belongs to any node at the bottom level, $p$ will be passed to the Spatial Dominance Test. Otherwise, $p$ will be discarded.

In the last step of spatial dominance test, we use the comparable region-based dominance test to reduce the cost of object comparison. All candidate points are compared in parallel. If one data point is not in the comparable region of any other data point, it will be eliminated immediately.

### 4.3 Multi-level IRG-based parallel filter

Based on multi-level independent region group and its properties (Sect. 4.1), we develop a novel multi-level IRG-based space partitioning scheme which partitions the search space into independent sub-spaces.

IRG-based space partitioning has been proposed for addressing spatial skyline queries in parallel. By utilizing the property of convex hull, only the points on the convex hull of query points are needed in query evaluation. This property helps to significantly reduce the cost of spatial dominance test; however, it also sets a limit to the parallelism of the solution because spatial skyline queries can be performed up to $|CH(Q)|$ processes/threads in parallel, where $|CH(Q)|$ indicates the number of points on the convex hull of query points $Q$. To fully utilize the computing power of GPUs, we propose an MIRG-based space partitioning scheme; every independent region can be further partitioned into smaller independent regions by overlapping it with an additional independent region group. Theoretically, the more the independent region groups are overlapped, the more independent regions can be used for query evaluation (see Property 9).

The correctness of multi-level IRG partitioning can be proven from the following two perspectives. (1) Sufficiency. Property 6 specifies the search space of spatial skyline queries, which covers all spatial skyline candidates. If an object is outside the search space, the object cannot be a spatial skyline because it must be spatially dominated by one of the pivot data points of $L^n IRG$. (2) Necessity. Property 8 describes a necessary object set for every candidate, which contains all objects with which the candidate must be compared.

Figure 6 displays an example of the partition tree, which represents the space partitioning of the example in Fig. 4. The root node at level 0 indicates the entire search space $R^d$. Three nodes at level 1 partition the search space into three sub-spaces by using $IRG(p_1, Q)$. The union of the three sub-spaces is the search space after applying $IRG(p_1, Q)$; the objects in one sub-space are independent from the ones in other two sub-spaces. Moreover, for every sub-space, any data point in the sub-space can be used to generate an IRG and partition the sub-space into smaller sub-spaces. Considering $IR(p_1, q_1)$ in Figs. 4a and 6 for example, $p_8$ is selected as a $L^2$ pivot in $IR(p_1, q_1)$, and $IRG(p_8, Q) = \{IR(p_8, q_1), IR(p_8, q_2), IR\ (p_8, q_3)\}$ is produced and overlapped with $IR(p_1, q_1)$. $IR(p_1, q_2) \in IR(p_8, q_2)$, $IR\ (p_1, q_3) \in IR\ (p_8, q_3)$, so those two $IR$s will not be partitioned. Thus, the node of $IR(p_1, q_1)$ has three child nodes, which are $IR\ (p_1, q_1) \cap IR\ (p_8, q_1)$, $IR\ (p_1, q_1) \cap IR\ (p_8, q_2)$ and $IR\ (p_1, q_1) \cap IR\ (p_8, q_3)$.

*Filter example.* Figure 4 displays an example of $L^2$ IRG-based parallel filtering, and Algorithm 2 describes the filter process in pseudocode. When Algorithm 2 visits the first level of the partition tree (Fig. 6), the three nodes at level

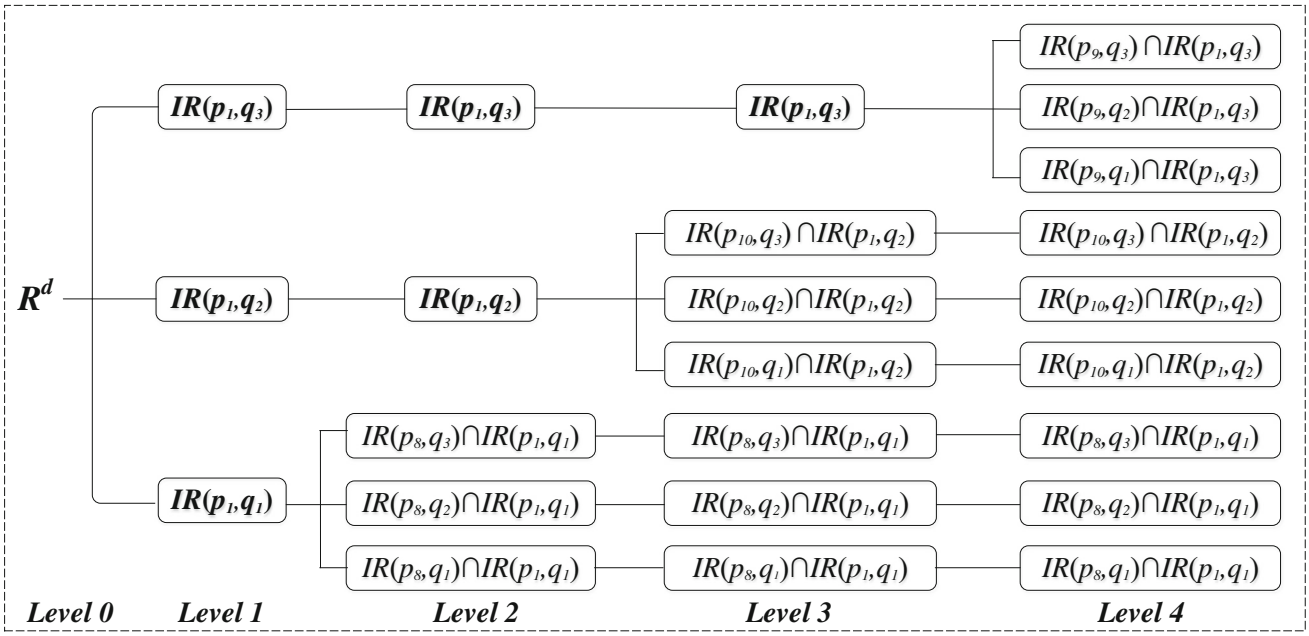


**Fig. 6** An example of the partition tree

**Algorithm 2** Parallel Filter

**Input:** $P$, $IRGs$
**Output:** $P'$

```
1: Function: ParallelFilter (P, IRGs)
2: P.dagger = 0;
3: for ∀p ∈ P (in parallel) do
4:     for ∀ ir ∈ L^m IRG (in parallel) do
5:         if p ∈ ir and p ∈ L^n IRG then
6:             p.dagger=1;
7: Use P.dagger to do Parallel Prefix Inclusive Scan & Repack;
8: return P';
9: EndFunction
```

one indicate the three sub-spaces partitioned by $L^1IRG$. All objects outside the three sub-spaces are eliminated (e.g., $p_7$). In this example, we highlight three sub-spaces in $IR(p_1, q_3)$ in red in Fig. 4a. Figure 6 level 2 nodes correspond to Fig. 4a $L^2$IRG independent regions. We can also observe that $L^4IRG$ ($|Q| + 1 = 4$) introduces 9 ($|Q|^2$) $IR$s at the bottom level. In summary, there are two ways to implement the parallel filter:

I. Execute bottom level nodes in parallel, which means that nine threads for each point will be generated in this specific example.
II. Pick a middle level $m$. Assign one thread for each node at level $m$, then traverse the child nodes at a higher level sequentially. If $m = 1$, we have three threads for each point in our example.

However, if we use $n$ threads for each point, we need $n$ bytes to store the threads' results for reduction. Using too many threads for each point would increase the reduction cost as well as GPU memory usage. We adopt the solution to parallelize the nodes in a middle level, and each thread traverses child nodes in a higher level sequentially. After all threads reach the bottom, we will execute a reduction and delete all filtered points.

In Algorithm 2, $P.dagger$ denotes whether data points $P$ are filtered. We use *parallel prefix inclusive scan* to calculate the repacked index in *P.index* from *P.dagger*, and shift spatial skyline candidates based on *P.index*. Parallel prefix inclusive scan is a standard GPU algorithm that computes a common sequential prefix scan in a parallel manner. For example, if we have a *P.dagger* that looks like [1, 0, 1, 1, 0] (1 indicates skyline candidate), a prefix inclusive scan will generate [0, 1, 1, 2, 3] (denoted as *P.index*), which is a linear scan that sums up all previous data. However, we cannot apply parallel prefix inclusive scan to our algorithm directly. Each position $i$ in the *P.index* array keeps the count of items that were not deleted before $i$. To take advantage of such an efficient algorithm, elements in *P.index* will be assigned with $-1$ if the corresponding element in *P.dagger* is 0. As a result, the revisited *P.index* is [0, $-1$, 1, 2, $-1$], which are the indices of skyline candidates (positive number).

### 4.4 *Spatial-GPU* Algorithm and implementation details

As the key component of *Spatial-GPU*, Sect. 4.3 has covered the filter process. In this section, we first introduce our pivot selection policy in Sect. 4.4.1. We then explain how to

**Algorithm 3** Calculating Pivots and Independent Regions

**Input:** $P$, $Q$
**Output:** $irs$

```
1: Function: CalculatingPivots&IR
2: τ = MAX_VALUE;
3: PI = ∅;                                   ▷ L^i pivots set
4: for ∀ idx ∈ {1,...,|Q|} (in parallel) do
5:   for ∀p ∈ P (in parallel reduction) do
6:     if idx == |Q| then
7:       if max_{q∈Q} D(p, q) < τ then
8:         pi = p ;                          ▷ L^1 pivot
9:         τ = max_{q∈Q} D(p, q) ;
10:    else
11:      q = Q[idx];
12:      if D(p, q) < D(PI[idx],q) then
13:        PI[idx] = p ;
14: Calculate all L^i IRG based on PI;
15: return L^1 IRG,...,L^n IRG;
16: EndFunction
```

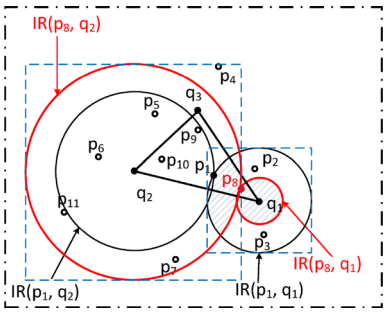


**Fig. 7** Simplified example of pivot selection

execute the *Spatial-GPU* algorithm with comparable region in Sect. 4.4.2. In Sect. 4.4.3, we introduce how to accelerate the dominance test with GPUs.

#### 4.4.1 Independent region pivot selection

*Spatial-GPU* utilizes the $L^n$ independent region group-based pre-filter method. The basic idea is constructing a global independent region group first, called $L^1IRG$. In every independent region that belongs to $L^iIRG(i = 1, ..., n-1)$, we pick an IRG pivot $p'$ and construct another independent region group. The overlap of $L^iIRG$ and $IRG(p', Q)$ is $L^{i+1}IRG$. Spatial skyline candidates must fall into at least one $IR$ ($IR \in L^{i+1}IRG$). The first step to implement this method is choosing optimal pivots that maximize the number of filtered data points as shown in Algorithm 3.

$L^1$ *pivot* is a global pivot that balances all independent regions. *Spatial-GPU* uses a parallel reduction to identify $L^1$ *pivot*, as the min of max distance value of each data point ( $L^1$ *pivot*=$\{p \mid min_{p\in P}\ max_{q\in Q}\ D(p,q)\}$ ).

$L^n$ *pivots* are local pivots that further partition and shrink lower-level IRGs. The purpose of $L^n$ IRG is to minimize the search region and maximize the partitions for parallelism.

In Fig. 7 as an example, the search region of $IR(p_1, q_1)$ decreases to $IR(p_1, q_1) \bigcap IRG(p_8, Q)$ (the shadow area). In Fig. 7, the point that minimizes the overlap region with $IR(p_1, q_1)$ should be a pivot for the next level.

#### 4.4.2 Comparable region-based spatial dominance test

In the process of parallel spatial dominance test (see Definition 1), we use *comparable region* to reduce the number of comparisons within the Euclidean distance metric. Note that comparable region is a portable approach, and can be decoupled from our solution. In this subsection, we will provide a formal definition of comparable region. For each point $p$, we only need to compare it with $p'$ that belongs to the same independent region. If $p$ and $p'$ are not in the same independent region, we can skip the expensive dominance test. With the same concept, for each point $p$, we only need to compare it with the $p'$ that belongs to the comparable region of $p$.

**Definition** Given a data point $p$ and $L^1$ pivot $pi$, and both $p$ and $pi$ are in $d$-dimensional Euclidean space $\mathbb{R}^d$. Assume $\tau = max_{q\in Q}D(pi, q)$. The *comparable region* of $p$ will be

$$\{p' \mid \wedge_{i=1}^{d}\ D(p.x_i - p'.x_i) \le 2\tau\}$$

For any spatial skyline candidate $p$, $p$ must be in at least one $IR$ ($IR \in L^1\ IRG$). We assume $p.IRs$=$\{IR \mid p \in IR\}$. The comparable region of $p$ must cover $p.IRs$.

For any data point $p$, we assume $p \in IR(pi,q_i)$. Based on the definition of independent region, $IR(pi,q_i)$=$\{l \mid D(l, q_i) \le D(pi, q_i)\}$. To reduce execution cost, the comparable region can be approximated by an MBR (Minimum Bounding Region), which is able to cover any $IR$. The sizes of comparable regions depend on the pivot and data points. Calculating the size of the comparable region for each point would inevitably cause considerable extra cost. In order to produce a global comparable region that is applicable on all data points without extra cost, the side length of MBR could be approximated to $2 * max_{q\in Q}\ D\ (pi,q)$.

The approximation may produce false positive. However, actual dominance tests will be conducted on all of the skyline candidates, which will guarantee the correctness of the final results. Because $L^1$ pivot is able to balance all $IR$s that belong to $L^1\ IRG$ (see Sect. 4.4.1), $\{D(pi, q) \mid \forall\ q \in Q\}$ has a low variance and the number of false positives is not large.

In the pivot selection phase, we already calculated $\tau = min_{p\in P}\ max_{q\in Q}D(p, q)$, and $pi = \{p \mid min_{p\in P}\ max_{q\in Q}\ D(p, q)\}$, which means $\tau = max_{q\in Q}D(pi,q)$. As a result, the side length of comparable region MBR = $2\tau$. Before every pair of dominance test between $p$ and $p'$, we only need to

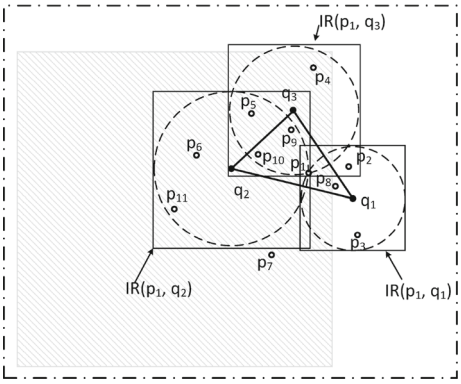


**Fig. 8** Comparable region-based BNL

**Algorithm 4** Parallel Spatial Skyline Operation

**Input:** $P'$, $Q$
**Output:** $ssky$
1: **Function: SpatialDominanceTest** ($P'$, $Q$)
2: $ssky = \emptyset$;
3: $P'.dagger = \emptyset$;
4: **for** $\forall p' \in P'$ (in parallel) **do**
5: **for** $\forall p'' \in P'$ (in parallel) **do**
6: **if** $|p.x - p'.x| > 2\tau \vee |p.y - p'.y| > 2\tau$ **then**
7: continue;
8: **else if** $p'$ is dominated by $p''$ **then**
9: $p'.dagger = 0$;
10: Final repack;
11: **return** $ssky$
12: **EndFunction**

check the following equation:

$$\vee_{i=1}^{d} D(p.x_i - p'.x_i) > 2\tau \tag{6}$$

If Eq. 6 holds, $p$ and $p'$ are incomparable.

Taking Fig. 8 as an example in $\mathbb{R}^2$, $\{p' \mid D(p_{11}.x - p'.x) \leq 2\tau \wedge D(p_{11}.y - p'.y) \leq 2\tau\}$ is the *comparable region* of $p_{11}$ (the shadow area). As we can see, for $p_{11}$ only, we avoid the dominance test between $p_{11}$ and $p_2$, $p_3$ and $p_8$.

As the outline of our algorithm described in Algorithm 1, after the parallel filter, we will execute parallel spatial skyline operations on candidate data points. The details of our spatial dominance operation are described in Algorithm 4, we use *P.dagger* to denote dominated data points (1 by default).

#### 4.4.3 Paralleled dominance test

As a simple representation, we use $|Q|$ to represent $|CH(Q)|$. $|Q|$ is the dimension of spatial dominance test. Instead of assigning the computation of each dominance test to one thread (used in [22]), we break down the work of checking all query points to $|Q|$ parts.

Intuitively, each thread works for one subset of data-points. However, there is a weakness for such solution. For example, if there is a thread named $A$, $\exists j,\ j \in [1, |Q|] \wedge D(p_1, Q[j]) < D(p_2, Q[j])$, thread $A$ will terminate in the middle of traversing from 1 to $|Q|$. However, if there is another thread named $B$ in the same warp, $\nexists j,\ j \in [1, |Q|] \wedge D(p_1, Q[j]) < D(p_2, Q[j])$. Thread $B$ will finish the traversing from 1 to $|Q|$, and terminate later than $A$. Because of the SIMD parallel nature of GPU, the overall computation will not benefit from early termination of $A$, and $B$ becomes the bottleneck. In order to improve GPU utilization and avoid thread divergence, we use $d$ threads for $d$-dimensional dominance tests.

As we know, threads are executed in warps. Thread divergence in a warp will not accelerate overall execution time. Our method parallelizes this comparison process, and each thread only compares one dimension. In this way, every dimension shares the same execution time.

For example, for each point, we create an array of size $|Q|$ called $flag$. We set $flag_i$ to 1 ($i = 1, ..., |Q|$) by default. If $p_i$ is dominated with respect to $q_j$ ($q_j \in Q$), we assign $flag_j = 0$. If $\vee_{j=1}^{|Q|} flag_j = 1$, $p_i$ is a spatial skyline candidate. In other words, we have to check whether $flag_j = 1$ for every $j$ ($j = 1, ..., |Q|$). In order to parallelize this process and eliminate thread divergence, each thread only checks one query point $q_j$, i.e., whether $p_i$ is dominated with respect to $q_j$. Each thread is only responsible to fill one $flag_j$. After filling the entire $flag$ array, we synchronize all threads in each thread block and execute a reduction. To reduce the time of final reduction in the final step, we store the flag array in *shared memory*. As we describe in the GPU overview, shared memory enjoys fast memory access, which is suitable for storing intermediate flag results. Paralleled dominance test can be treated as a plug-in. The choice of adopting this plug-in or not depends on the compatibility with the hardware structure of the GPU.

### 4.5 Comparison with *SkyAlign*

In this section, we compare our approach with state-of-the-art GPU-based skyline algorithm *SkyAlign* [22] from two perspectives: filter process acceleration and thread divergence. We will show that our multi-level IRG-based filter eliminates more non-candidates with high efficiency. *Spatial-GPU* also takes advantage of GPU multi-threading scheme with low divergence.

#### 4.5.1 Filter process acceleration

*SkyAlign* applies a threshold-based filter, which eliminates data points that have any values smaller than the threshold or all values equal to the threshold. The threshold $\tau$ is set as the min of max values and the filtering process can be performed in parallel. However, the filtering method cannot be parallelized along dimensions. In [22], to filter each point

$p$, one thread will check $D(p, Q[j]) > \tau(j = 1, ..., |Q|)$ sequentially, which is $O(|Q|)$. This pre-filter method has a major drawback when applying to spatial skyline. A spatial skyline problem commonly has multiple query points, which results in relatively high-dimensional input data after transferring to general skyline problem. Parallel mechanism in this method is not applied along dimensions, which will not fully utilize GPU cores.

However, in our filter process, we take advantage of the spatial property of input data. Instead of transferring spatial query points to distance attributes, we store the coordinates of input data points. In the filter process, each thread is only responsible for checking whether a data point $p$ is in the MBR of the independent region, which is $O(1)$.

Compared with *SkyAlign*, the proposed multi-level IRG-based filtering method is able to not only perform filtering with less run-time, but also further eliminate a portion of data points because of higher-level IRGs. Specifically, after pivots are selected and the partition tree is generated, our method visits a certain level of the partition tree in parallel and prunes the data points by detecting whether they are in any node of the tree. We also conduct optimization when the number of query points $|Q|$ becomes too large. For example, if $|Q| = 10$, level $|Q|+1$ of the partition tree will execute $|Q|*|Q| = 100$ threads for each point, which causes the reduction cost exceeding parallel benefit. In this case, we need to parallelize the middle level of the partition tree instead of the bottom level. In general, at level $m$, we start $|m|$ ($|m|$ is the number of nodes at level $m$) threads in parallel, each of which receives a node (or a partition) at level $m$ of the tree. If a data point is not in any of the nodes at level $m$, the data point can be eliminated because it is not in the $L^m IRG$ (Property 6). After all the unqualified data points are tagged, we proceed to scan the remaining data points in parallel with respect to all the child nodes at a higher level sequentially. In this way, we further eliminate non-candidates without extra parallel resources.

#### 4.5.2 Low divergence

*SkyAlign* uses point-based recursively partitioning methods to induce a quad-tree partitioning for the dataset, and records skyline points when found in the tree. The recursively defined point-based methods are not suitable for the branching sensitive GPU architecture due to high divergence. Even though *SkyAlign* uses the global alignment to split the quad-tree, the results of mask tests diverge sporadically. We will test this in Sect. 6.3.1. The global partitioning scheme means that it is inevitable to sort all the points along *every* dimension. Moreover, finding a skyline point in the previous method requires traversing the tree, which cannot take advantage of parallel schemes.

Our *Spatial-GPU* uses an independent region-based multi-level filtering method (Sect. 4.3). The key algorithmic idea is taking the advantages of parallel multi-core mechanism and reducing single-threaded sequential processes. Most existing skyline algorithms suffer from sorting process. For example, *SkyAlign* uses GPU radix sort on each dimension independently. If we have 10 query points in spatial skyline and only one GPU, 10 times sequential sorts are inevitable. That explains why our algorithm outperforms *SkyAlign*.

In this section, we propose our advanced parallel spatial skyline solution using GPU computing platform. First of all, we propose a new concept of multi-level independent region group, and briefly present the framework of the solution. After a multi-level independent-region-based pre-filter is developed, our spatial skyline algorithm is illustrated in detail in Sect. 4.4. Finally, we present the difference between *SkyAlign* and our algorithm.

## 5 Spatial skyline evaluation using map-reduce scheme

In this section, we propose our advanced parallel spatial skyline solution using MapReduce for scenarios of large-scale datasets. First of all, we briefly present the framework of the solution. Then, our spatial skyline algorithm is illustrated in detail in Sect. 5.2. The concepts of independent regions and pruning regions are introduced to optimize the process of spatial skyline evaluation. Finally, we discuss three critical implementation issues in our solution.

### 5.1 Framework overview

Our solution consists of three MapReduce phases, which receive a set of data points $P$ and a set of query points $Q$ as inputs, and output spatial skyline points of $P$ with respect

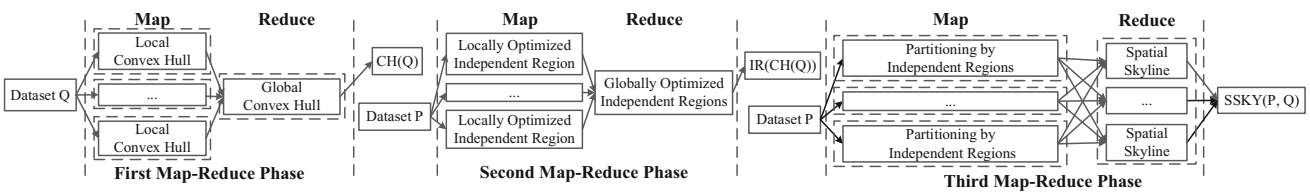


**Fig. 9** An overview of the parallel spatial skyline processing using MapReduce

to $Q$. As illustrated in Fig. 9, we calculate the convex hull of $Q$ in the first MapReduce phase. $Q$ is initially partitioned into subsets of equal size, and each map function finds the local convex hull of query points in a subset. Then, a reduce function generates the global convex hull of $Q$ by merging the local convex hulls. Convex hull algorithms like Graham scan could be employed in each map and reduce function [23]. Due to high complexity of convex hull computation, a filtering method can be used to filter out unqualified points with lower cost. For example, Eldawy et al. observed that the convex points must be at least one of four types of skyline points of $Q$ (max-max, min-max, max-min and min-min) in a 2-dimensional space, and applied skyline algorithms as a filtering step in their CG_Hadoop system [24].

An intuitive spatial skyline method requires examination of the spatial dominance between every pair of data points. Sharifzadeh and Shahabi utilized the R-tree and Voronoi diagram as indices in their $B^2S^2$ and $VS^2$ algorithms [9]. Son *et al.* enhanced $VS^2$ by reducing the number of dominance tests [10]. However, extending these methods to a parallel solution is non-trivial. Designing an algorithm that efficiently maintains indices over data in a distributed and/or parallel environment requires expertise and extensive experience. To address this issue, we take advantage of independent regions, in each of which spatial skyline points do not depend on any data points outside the independent region. With the independence, the input data points can be partitioned by their independent regions, and spatial skyline points can be calculated in parallel. Therefore, after the completion of convex hull computation, we calculate the independent regions based on the convex hull and the input data points $P$ in the second phase. Each map function takes a subset of $P$ and the convex hull of $Q$ as inputs and outputs a locally optimal independent region pivot (see Fig. 2). Then a reduce function produces a globally optimal independent region pivot by merging the intermediate results. More details of independent region pivot selection will be discussed in Sect. 5.5.1.

In the third phase, $P$ is initially partitioned, and each map function finds the independent regions of data points in a split. The output of the map functions can be represented as $< IR.id, p >$, where $IR.id$ denotes the unique identifier of the independent region associated with a data point $p$. There are three possible cases: (1) data points are eliminated if they are outside all independent regions, (2) data points are marked and output as spatial skylines by mappers and reducers if they are inside the convex hull of $Q$. These data points are needed in reduce functions, because they may spatially dominate data points in category 3, and (3) data points are produced with their associated independent regions if they fall in at least one independent region. These data points will be processed by reducers to find spatial skylines in the independent regions. If a data point is inside two or more independent regions, the map function will produce a pair of $< IR.id, p >$ for every associated independent region. After the shuffle phase, data points in $P$ are grouped by independent regions and sent to reduce functions for spatial skyline calculation in parallel. Finally, the global spatial skyline points are the union of the output of reduce functions. A data point could be associated with two or more independent regions, which may introduce duplicates in the results. We design an elimination method to remove duplicates in our solution. The method will be presented in Sect. 5.5.3.

Figure 2 shows an example of spatial skyline query over three query points and eight data points ($Q$=$\{q_1, q_2, q_3, q_4\}$, $P$=$\{p_1, ..., p_8\}$). First of all, the convex hull of query points ($CH(Q)$) is generated in the first MapReduce phase. Then, the globally optimal independent region pivot is found by using $P$ and $CH(Q)$ in the second MapReduce phase. Each mapper takes a split of $P$ and $CH(Q)$ (a constant global variable) and selects a local optimal independent region pivot, and a reducer outputs the globally optimal pivot. In the third MapReduce phase, each mapper receives a split of $P$, and the pivot and $CH(Q)$ (as two constant global variables), and produces object points with their associated independent regions. In the example, there are three independent regions ($\{IR(p_1, q_1), IR(p_1, q_2), IR(p_1, q_3)\}$). All the independent regions can be calculated from the pivot and $CH(Q)$ in mappers. Moreover, object points are associated with the independent regions where they locate in. If we use $ir_1$, $ir_2$ and $ir_3$ to denote $IR(p_1, q_1)$, $IR(p_1, q_2)$ and $IR(p_1, q_3)$, then $p_1$ is associated with $ir_1$, $p_5$ is associated with $ir_2$ and etc. After the shuffle phase, $< ir_1, p_1 >$, $< ir_1, p_2 >$, $< ir_1, p_3 >$ and $< ir_1, p_8 >$ are grouped and sent to the first reducer, $< ir_2, p_1 >$, $< ir_2, p_5 >$, and $< ir_2, p_6 >$ are passed to the second reducer, and $< ir_3, p_1 >$ $< ir_3, p_4 >$ $< ir_3, p_5 >$ are processed in the third reducer. In this case, $p_1$ is a special object point, which is in all three independent regions. As we will discuss our elimination method in Sect. 5.5.3, $p_1$ will only be output by the first reducer. Thus, the first reducer outputs $p_1$, $p_2$ and $p_8$ as spatial skylines and discards $p_3$ because it is dominated by $p_8$. The second reducer produces $p_5$ and $p_6$. The third reducer does not output any object because $p_4$ is eliminated in the spatial dominance test and $p_5$ has been produced in the second reducer. Finally, the result of the spatial skyline query is the union of the results of reducers, which is $\{p_1, p_2, p_5, p_6, p_8\}$.

## 5.2 Spatial skyline calculation

In the second and third MapReduce phases, we generate independent regions based on the convex hull of $Q$ and a set of data points $P$ for spatial skyline computation in parallel. Due to high cost of the spatial dominance test that requires comparing the distance from data points to all convex points, we introduce pruning regions in independent regions. A prun-

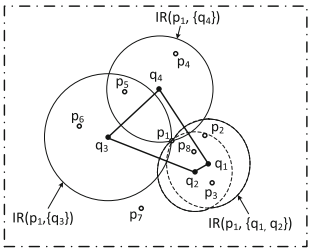


**Fig. 10** An example of merged independent regions in $\mathbb{R}^2$

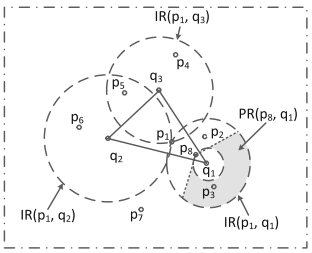


**Fig. 11** An example of pruning regions in $\mathbb{R}^2$

ing region can be defined by a data point inside $CH(Q)$, a convex point and its adjacent convex points. If a data point is in a pruning region, the point can be discarded without the dominance test.

The independent regions are determined by the independent region pivot and the convex hull of $Q$. The convex hull is uniquely determined by input query points $Q$; however, theoretically, the pivot can be arbitrarily selected. Since the independent regions specify the search region that contains spatial skyline candidates, an intuitive strategy of the data point selection is to select the data point that minimizes the total volume of its independent regions (Fig. 10).

Figure 2 displays an example that utilizes independent regions in spatial skyline evaluation in $\mathbb{R}^2$. The datasets $P$ and $Q$ consist of 8 data points and 4 query points, respectively. The convex points of the convex hull of $Q$ are $q_1$, $q_2$ and $q_3$. The three dashed circles indicate three independent regions generated by $p_1$ and the convex points. In this example, $P$ is partitioned into three subsets, which are $P_1 = \{p_1, p_2, p_3, p_8\}$, $P_2 = \{p_1, p_4, p_5\}$ and $P_3 = \{p_1, p_5, p_6\}$. Because $p_1$ and $p_8$ are in the convex hull, they are spatial skylines [9]. $p_7$ is outside all independent regions and can be discarded by mappers in the third phase. $p_5$ is in $IR(p_1, q_2)$ and $IR(p_1, q_3)$, thus $p_5$ is associated with both independent regions. Then, the spatial skylines in independent regions are calculated independently. Figure 11 shows an example of the pruning region in $IR(p_1, q_1)$ (the formal definition of pruning regions will be presented in Sect. 5.3). $p_8$ is a data point that is closer to $q_1$ than $p_1$. Thus, we create a pruning region $PR(p_8, q_1)$ (highlighted in gray) in $IR(p_1, q_1)$ to filter out data points dominated by $p_8$. In the example, $p_3$ falls in $PR(p_8, q_1)$ and can be discarded without being compared with $p_2$. Thus, $p_2$ is the only data point requiring spatial dominance test, comparing its distance to all convex points with the one of $p_8$. Our spatial skyline algorithm will be presented in Sect. 5.4.

### 5.3 Pruning regions in independent regions

In the third MapReduce phase, a reduce function calculates spatial skylines of a set of data points in an independent region. In particular, the data points are comparing their distances to all convex points of $CH(Q)$ with all other data points (spatial skylines do not depend on non-convex points [9]), and the ones are discarded if they are spatially dominated in the same independent region. The data point comparison would be expensive when the number of convex points of $CH(Q)$ becomes large. Thus, to minimize the cost of the dominance test, we propose a pruning method that is able to efficiently filter out unqualified data points without accessing all convex points of $CH(Q)$. This method defines a pruning region in each independent region. If data points fall in the pruning region, they can be discarded because there must exist a data point dominating these data points. We will first illustrate the pruning regions in a 2-dimensional space and then provide a formal definition and proof of the pruning regions in high-dimensional spaces.

Figures 12 and 13 show an example of a pruning region in $\mathbb{R}^2$. Given a query point $q$, two data points $p$ and $v$, let $L_{qx}$ be a line connecting $q$ to any point $x \in \mathbb{R}^2$, and $L_{vq}$ be the line of $q$ and $v$. We build a 2-dimensional Cartesian coordinate system, in which $q$ is the origin and $L_{qx}$ is $x$ axis. $L_{qx}$ and $L_{vq}$ partition $\mathbb{R}^2$ into two half-spaces, denoted by $S^-_{qx}$ and $S^+_{qx}$, and $S^-_{vq}$ and $S^+_{vq}$, respectively.

**Theorem 2** *If $p$ and $v$ satisfy*

$$\begin{aligned} &(1)\ v \in S^+_{qx} \text{ and } p \in S^-_{qx} \\ &(2)\ v.x \le p.x \\ &(3)\ D(v,q) > D(p,q) \end{aligned} \tag{7}$$

*then $p$ spatially dominates $v$ with respect to any point $q^*$, $q^* \in S^-_{qx} \bigcap S^+_{vq}$.*

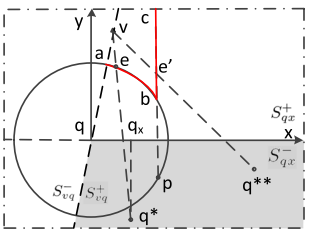


**Fig. 12** A pruning region using $p$ and a visible facet $L_{qx}$

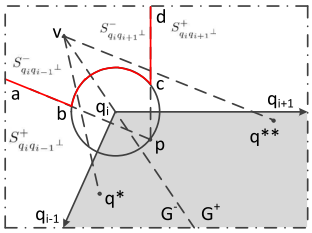


**Fig. 13** An example of $PR(p, q_i)$

***Proof*** As a case of $p.x \geq 0$ shown in Fig. 12, the three conditions indicate that (1) $L_{qx}$ partitions $v$ and $p$ into two half-spaces, $v.y > 0$ and $p.y < 0$; (2) if we create a line $L_{pc}$ perpendicular to $L_{qx}$, then $v$ is at the left side of $L_{pc}$; (3) $v$ is outside of the circle centered at $q$ with radius $D(p, q)$. The circle intersects with $L_{pc}$ at $p$ and $b$.

Given any data point $q^*$ or $q^{**}$ in $S^-_{qx} \bigcap S^+_{vq}$ (highlighted in gray), $L_{vq^*}$ must intersect with either an arc from $a$ to $b$ ($arc_{ab}$) or a line from $b$ to $c$ ($L_{bc}$). If $L_{vq^*}$ intersects with $arc_{ab}$ at $e$, then we can get $D(p, q) = D(e, q)$ and $p.x > e.x$. Given a query point $q_x = \{q^*.x, 0\}$ on $L_{qx}$, then

$$\begin{aligned} D(e, q_x) &= \sqrt{(e.x - q^*.x)^2 + (e.y)^2} \\ &= \sqrt{D(e, q)^2 - 2 \cdot (e.x) \cdot (q^*.x) + (q^*.x)^2} \\ &> \sqrt{D(p, q)^2 - 2 \cdot (p.x) \cdot (q^*.x) + (q^*.x)^2} \\ &= \sqrt{(p.x - q^*.x)^2 + (p.y)^2} \\ &= D(p, q_x) \end{aligned} \tag{8}$$

thus, $D(p, q_x) < D(v, q_x)$. If $q_x$ is moved to $q^*$ ($q^*.y < 0$), then $D(p, q^*) < D(v, q^*)$ is also held. On the other hand, if $L_{vq^{**}}$ intersects with $L_{bc}$ at $e'$, then $D(p, q^{**}) < D(e', q^{**})$, because $L_{qx}$ is the bisector line of $p$ and $b$, and both $p$ and $q^{**}$ are in $S^-_{qx}$. Thus, $D(p, q^{**}) < D(e', q^{**}) \leq D(v, q^{**})$. We can get the similar result in the case of $p.x < 0$. Therefore, $p$ spatially dominates $v$ with respect to any query point in $S^-_{qx} \bigcap S^+_{vq}$. □

In a two-dimensional space, a convex hull is a convex polygon, in which each convex point has two adjacent convex points. Figure 13 shows three convex points $q_{i-1}, q_i$, and $q_{i+1}$ of a convex hull $CH(Q)$. $q_{i-1}$ and $q_{i+1}$ are adjacent to $q_i$. Line segments $L_{q_i q_{i-1}}$ and $L_{q_i q_{i+1}}$ are two visible facets from a data point $v$ outside $CH(Q)$ [25].

**Theorem 3** *In a two-dimensional space, given a query point $q_i \in CH(Q)$ and a data point $v$ outside $CH(Q)$, let $A^{\triangle}_{q_i}$ be a set of adjacent convex points of $q_i$, and $p$ be an invisible data point from $v$. Each of the lines from $p$ perpendicular to $L_{q_i q_j}$ ($q_j \in A^{\triangle}_{q_i}$) partitions the space into two closed half-spaces. Let $S^-_{q_i q_j \perp}$ be the half-space containing $q_i$. Then, any data point $v$ outside $CH(Q)$ satisfying*

$$\begin{aligned} &(1)\ v \in S^-_{q_i q_j \perp},\ q_j \in A^{\triangle}_{q_i} \\ &(2)\ D(v, q_i) > D(p, q_i) \end{aligned} \tag{9}$$

*is spatially dominated by $p$ with respect to $Q$.*

***Proof*** In Fig. 13, $L_{ab}$ and $L_{cd}$ are two lines from $p$ perpendicular to $L_{q_i q_{i-1}}$ and $L_{q_i q_{i+1}}$, respectively. $L_{ab}$, $L_{cd}$ and $arc_{bc}$ separate $v$ from the convex hull $CH(Q)$. $L_{vq_i}$ partitions $CH(Q)$ into two closed half regions, $G^-$ and $G^+$; all convex points are in either $G^-$ or $G^+$. If a convex point $q^*$ is in $G^-$, we can easily get $D(p, q^*) < D(v, q^*)$ by using Theorem 2. The similar result can be obtained in the case that any convex point $q^{**}$ is in $G^+$. Thus, $v$ is spatially dominated by $p$ with respect to $Q$. □

After an illustration of pruning regions in $\mathbb{R}^2$, we extend the concept of pruning regions to high-dimensional spaces. In a $d$-dimensional space, a convex hull of query points $Q$ can be represented by a set of half-spaces $\mathcal{H}$, where $CH(Q) = \bigcap_{h^+ \in \mathcal{H}} h^+$. The bisector hyper-plane of each half-space contains a *(d-1)*-dimensional facet of the convex hull, which can be determined by a convex point and a subset of its adjacent convex points. The formal definition of the pruning regions is provided as follows.

**Definition 7** (*Pruning Regions*) In a $d$-dimensional space, given a convex hull of query points $CH(Q)$, a data point $v$ outside $CH(Q)$, a visible convex point $q$ and an invisible data point $p$ from $v$, let $A^{\triangle}_q$ be a set of adjacent convex points of $q$ in facets, $h^{\perp}_{qq_j}$ be the *(d-1)*-dimensional hyper-plane that contains $p$ and is perpendicular to $L_{qq_j}, q_j \in A^{\triangle}_q$. $h^{\perp}_{qq_j}$ partitions the space into two closed half-spaces. The one containing $q$ is denoted by $S^-_{h^{\perp}_{qq_j}}$. Then any data point $v$ outside $CH(Q)$ satisfying

$$\begin{aligned} &(1)\ v \in S^-_{h^{\perp}_{qq_j}},\ q_j \in A^{\triangle}_q \\ &(2)\ D(v, q) > D(p, q) \end{aligned} \tag{10}$$

is spatially dominated by $p$ with respect to $Q$. The region containing all possible $v$ is called a pruning region of $p$ and $q$, denoted by $PR(p, q)$.

***Proof*** The proof is by induction. The pruning region in a *2*-dimensional space is proven in Theorem 3. We assume that the pruning region is held in an *(i-1)*-dimensional space ($i \geq 3$), then, in an $i$-dimensional space, since $v$ is outside $CH(Q)$, the line connecting $v$ with any convex point $q^*$ must intersect with a visible closed *(i-1)*-dimensional facet $F$ of $CH(Q)$. The hyper-sphere centered at $q$ with radius $D(p, q)$ and $h^{\perp}_{qq_j}$ ($q_j \in A^{\triangle}_q$) separate $v$ from $CH(Q)$. If a ray from

$v$ to $q^*$ intersects with the hyper-sphere at $e$ earlier than any $h^{\perp}_{qq_j}(q_j \in A^{\triangle}_q)$, then given any convex point $q_k \in A^{\triangle}_q$, we can build an $i$-dimensional Cartesian coordinate system, in which $F$ is a hyper-plane ($x_i = 0$), $v.x_i > 0$, $p.x_i < 0$ and $L_{qq_k}$ ($L_{qq_k} \in F$) is $x$ axis. Any query point on $L_{qq_k}$ can be represented by $\{x_1, 0, ..., 0\}$. Since $D(e, q) = D(p, q)$ and $e.x_1 < p.x_1$, given any query point $q_x$ on $L_{qq_k}$, we can get

$$\begin{aligned} D(e, q_x) &= \sqrt{(e.x_1 - q_x.x_1)^2 + D(e, L_{qq_k})^2} \\ &= \sqrt{D(e, q)^2 - 2 \cdot (e.x_1) \cdot (q_x.x_1) + (q_x.x_1)^2} \\ &> \sqrt{D(p, q)^2 - 2 \cdot (p.x_1) \cdot (q_x.x_1) + (q_x.x_1)^2} \\ &= \sqrt{(p.x_1 - q_x.x_1)^2 + D(p, L_{qq_k})^2} \\ &= D(p, q_x) \end{aligned} \tag{11}$$

where $D(e, L_{qq_k})$ denotes the distance from $e$ to the line $L_{qq_k}$. Thus, we can get that, given any query point $q'$ satisfying $\mathrm{D}(p, q') \leq \mathrm{D}(e, q')$, if $q'$ is moved to $q''$ on any of the directions from $q$ to its adjacent convex points in $F$, $\mathrm{D}(p, q'') < \mathrm{D}(e, q'')$ is also held. Therefore, $p$ is closer to any query point in $F$ than $e$. Since, $v.x_i > 0$, $p.x_i < 0$ and $q^*.x_i < 0$, so $D(p, q^*) < D(v, q^*)$. On the other hand, if a ray from $v$ to $q^*$ first intersects with $h^{\perp}_{qq_k}$ at $e'$, let $q^{\circledast}$ be the center of the intersection of $h^{\perp}_{qq_k}$ and the hyper-sphere centered at $q$ with radius $\mathrm{D}(p, q)$, $h^{\perp}_{qq_k}$ is an *(i-1)*-dimensional hyper-plane, in which $D(e', q^{\circledast}) > D(p, q^{\circledast})$, and $e' \in S^{-}_{h_{qq_t}}, q_t \neq q_k, q_t, q_k \in A^{\triangle}_q$, which satisfies the conditions in an *(i-1)*-dimensional space. Thus, $D(p, q^{\circledast}) < D(e', q^{\circledast})$; then $D(p, q^*) < D(e', q^*) \leq D(v, q^*)$. Therefore, this concludes the proof. □

Figure 14 shows an example of the convex hull of query points $Q$ in a 3-dimensional space. $v$ is a data point outside

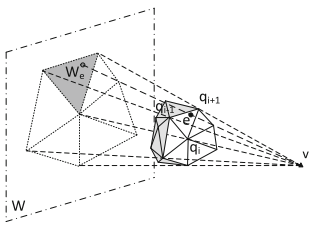


**Fig. 14** An example of visible facets of a convex hull in a 3-dimensional space

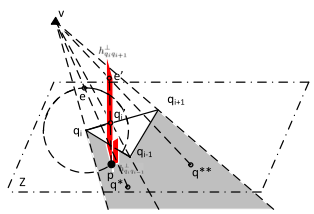


**Fig. 15** An example of a visible facet after a coordinate transformation

the convex hull. The line $L_{vw_e}$ intersects with a visible facet $F$ at $e$. $F$ can be determined by three convex points $q_{i-1}$, $q_i$ and $q_{i+1}$. After a coordinate transformation, $F$ is transformed to be on plane $Z$ ($z = 0$), $v.z > 0$ and $p.z < 0$, as displayed in Fig. 15. $L_{q_iq_{i+1}}$ is the $x$ axis in plane $Z$. The area invisible from $v$, including $p$, is highlighted in gray. Two hyper-planes $h^{\perp}_{q_iq_{i-1}}$ and $h^{\perp}_{q_iq_{i+1}}$ perpendicular to $L_{q_iq_{i-1}}$ and $L_{q_iq_{i+1}}$ are highlighted in red. $q_{i+1}$ and $q_{i-1}$ are two elements of $A^{\triangle}_{q_i}$. If a ray from $v$ to $q^*$ first intersects with the sphere centered at $q_i$ with radius $D(p, q_i)$ at $e$, then according to Eq. 11, we can get that given any point $q'$ on $L_{q_iq_{i+1}}$, $q'.x \geq q_i.x$, $D(p, q') \leq D(e, q')$, and moving the point on the direction from $q_i$ to $q_{i+1}$ with distance $\triangle d$ ($\triangle d > 0$) makes the point closer to $p$ than $e$. The similar result can be obtained on the direction from $q_i$ to $q_{i-1}$. Thus, any query point in the facet $F$ is closer to $p$ than $e$. Moreover, $e.z > 0$, $q.z < 0$ and $q^*.z < 0$, we can get that $D(p, q^*) < D(e, q^*) < D(v, q^*)$. On the other hand, if a ray from $v$ to $q^{**}$ first intersects with $h^{\perp}_{q_iq_{i+1}}$ at $e'$, then $e'.x = p.x$, and $D(e', q_i) > D(p, q_i)$. Let $q'_i$ be the intersection of $L_{q_iq_{i+1}}$ and $h^{\perp}_{q_iq_{i+1}}$, $D(e', q'_i) > D(p, q'_i)$. $h^{\perp}_{q_iq_{i-1}}$ intersects with $h^{\perp}_{q_iq_{i+1}}$ at a line, which contains $p$ and partitions $h^{\perp}_{q_iq_{i+1}}$ into two closed half-spaces. $q'_i$ and $e'$ are in the same half-space. By Theorem 3, $D(e', q'_i) > D(p, q'_i)$. Therefore, $D(v, q^{**}) \geq D(e, q^{**}) > D(p, q^{**})$, and $v$ is spatially dominated by $p$ with respect to $Q$.

### 5.4 Spatial skyline algorithm

With the concept of pruning regions, we present our spatial skyline algorithm used in reduce functions of the third phase. The input data points are grouped by their independent regions through the shuffle phase, and unqualified data points outside independent regions have been discarded in map functions. The fundamental idea of our method is to first eliminate data points by using pruning regions. If they are not in any pruning region, they are needed to compare with all other potential spatial skyline candidates for spatial dominance test.

The details of our method are described in Algorithm 5. The algorithm receives all data points in an independent region $IR(p, q_i)$, denoted by $P_i$, and the convex hull of query points $Q$. We use $chsky$ and $lssky$ to keep local spatial skylines inside and outside $CH(Q)$, respectively. $PR$ abstracts pruning regions of the spatial skyline candidates. The union of $chsky$ and $lssky$ are output as spatial skylines in the independent region, which is a subset of the global spatial skylines of the query.

In particular, the algorithm first finds all the data points in $CH(Q)$. These data points are kept in $chsky$, and used to build pruning regions $PR$ (from lines 4 to 9). $lssky$ temporarily maintains all data points outside $CH(Q)$. Then, each data point in $lssky$ is visited for the dominance test

**Algorithm 5** Spatial Skyline Algorithm

**Input:** $P_i, CH(Q)$
**Output:** $lssky \cup chsky$

```
1: lssky = ∅;
2: chsky = ∅;
3: PR = ∅;
4: for ∀p ∈ P_i do
5:     if p is inside CH(Q) then
6:         chsky = chsky ∪ {p};
7:         PR = PR ∪ PR(p, q_i);
8:     else
9:         lssky = lssky ∪ {p};
10: for ∀p ∈ lssky do
11:     if p is in PR then
12:         lssky = lssky - {p};
13:         Continue;
14:     if ∃ p' ∈ (chsky ∪ lssky), p' ≠ p, p' ≺^Q p then
15:         lssky = lssky - {p};
    return lssky ∪ chsky;
```

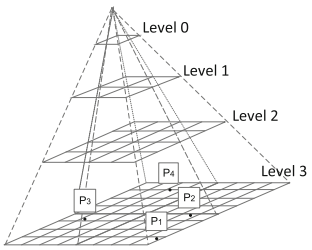


**Fig. 16** An example of $Grid(lssky \cup chsky)$

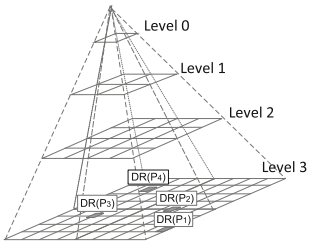


**Fig. 17** An example of $Grid(DR(lssky \cup chsky))$

(lines 10 to 15). If a data point falls in any pruning region, the data point will be removed from $lssky$. If the data point is outside the pruning regions, it needs to compare with all other data points in $chsky$ and $lssky$ and will be eliminated if it is dominated.

To minimize the cost of the dominance test in line 14, we use two multi-level grids, $Grid\ (lssky \cup chsky)$ and $Grid\ (DR(lssky \cup chsky))$, to maintain spatial skyline candidates and their dominator regions (defined in Sect. 2.1). The two grids are always synchronized; once there is a data point inserted into or removed from $Grid\ (lssky \cup chsky)$, $Grid\ (DR(lssky \cup chsky))$ is updated accordingly. Figures 16 and 17 display an example of the two grids. The cells at the bottom level keep the references of spatial skyline candidates and their dominator regions. Parent cells maintain the proximity information of their child cells. In the dominance test of a new data point $p$, we first check if $p$ is dominated by other data points. We calculate the dominator region of $p$ and visit $Grid\ (lssky \cup chsky)$ from top to bottom to see if there is a data point falling in the dominator region. The iteration can stop at any intermediate level when either of the two conditions is satisfied: (1) all cells intersecting with the dominator region do not contain any data point ($p$ is not dominated by spatial skyline candidates in $lssky \cup chsky$); (2) a cell inside the dominator region contains a data point ($p$ is dominated by the data point). If $p$ is not dominated, then we visit $Grid\ (DR(lssky \cup chsky))$ in a similar manner to see if $p$ dominates any data point in $lssky \cup chsky$. If $p$ falls in the dominator region of $p'$, then $p'$ and its dominator region will be removed from both $Grid\ (lssky \cup chsky)$ and $Grid\ (DR(lssky \cup chsky))$.

## 5.5 Implementation issues

In this subsection, we discuss three implementation issues in our solutions.

### 5.5.1 Independent region pivot selection

In the second MapReduce phase, the search space is partitioned into a number of independent regions. The spatial skylines are calculated in parallel by reducers in the third MapReduce phase. The execution time of a parallel program is determined by the slowest process, and the spatial skyline algorithm takes longer on larger inputs. Thus, distributing the data points to reducers in a balanced manner is critical to our approach.

If the data points are uniformly distributed in the search space, the number of data points in an independent region is proportional to the volume of the independent region, which depends on the distance between the independent region pivot and the convex point. Theoretically, the point with equal distance to all convex points is the optimal independent region pivot, which could split data points in equal size. However, the optimal pivot may not exist in irregular convex hull. Moreover, the point that minimizes the total volume of independent regions would be an alternative optimal pivot. Nevertheless, the cost of finding the point is expensive.

Thus, we turn to an approximation method in our implementation. After the convex hull is calculated, we choose the center of the Minimum Bounding Rectangle (MBR) of the convex hull as the independent region pivot.

### 5.5.2 Independent region merging

In the third phase of our solution, a reducer processes data points in an independent region. The number of reducers needed in the spatial skyline calculation depends on the number of independent regions or the number of convex points

in the convex hull of query points $Q$. Since the size of the convex hull would be large, the task maintenance and communication overhead in MapReduce framework would be unacceptably high.

Thus, there are two merging strategies that can be applied to our proposed solution if the number of independent regions is much greater than the number of available computing resources. In the strategies, we assume that objects are uniformly distributed in the search space. The smaller the total volume of independent regions is, the less the number of objects are processed in spatial skyline computation.

*Shortest distance merging* In this method, we merge the closest pair of two neighboring independent regions. The distance of two independent regions is evaluated by the distance between the centers of the independent regions. We assume that there is a higher possibility that two independent regions overlap with each other if they are close. Merging two overlapped independent regions may reduce the cost of spatial skyline computation for the following two reasons: (1) the objects in the overlapping region are fed to one reducer instead of two, which minimizes the total number of objects in the spatial dominance test; (2) the pruning regions of the independent regions are also merged; more objects could be eliminated without the dominance test. Take Fig. 10 for example, $q_1$ and $q_2$ are the closest pair of convex points in the figure. $IR(p_1, q_1)$ and $IR(p_1, q_2)$ are merged, and the new independent region is denoted by $IR(p_1, \{q_1, q_2\})$. So, $p_3$ and $p_8$ are only processed by the reducer, which receives $IR(p_1, \{q_1, q_2\})$. The pruning region of $IR(p_1, \{q_1, q_2\})$ is $\mathrm{PR}(p_1, q_1) \cup \mathrm{PR}(p_1, q_2)$.

In our implementation, we iterate the convex hull in counter clockwise order, and calculate the distance between every pair of two consecutive independent regions. Let $n$ be the number of computing resources available to the spatial skyline evaluation and $m$ be the number of convex points, we will merge the top $m - n$ ($m \geq n$) closest pairs of the independent regions (the number of pairs of independent regions is equal to the number of convex points).

*Threshold-based merging* An alternative strategy merges independent regions by considering the volume of the overlapping region of two independent regions. In this method, we visit the independent regions in counter clockwise order. Given two consecutive independent regions, we calculate the ratio of volume of the overlapping region of the two independent regions to the volume of the smaller independent region. If the ratio is higher than a specific threshold, the two independent regions will be merged. Another difference from the first method is that two or more independent regions may be merged if they are close to each other. The ratio can be defined as follows.

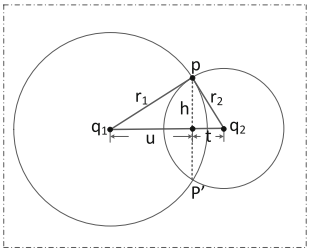


**Fig. 18** Volume of overlapping region of two independent regions

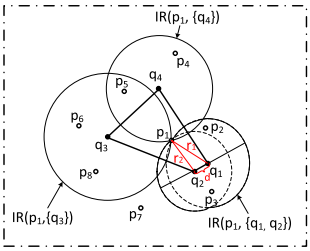


**Fig. 19** An example of independent regions in a 2-dimensional space

$$\mathrm{ratio}(q_1, q_2) = \frac{\mathrm{Vol}^d(IR(p_1, q_1)) \cap \mathrm{Vol}^d(IR(p_1, q_2))}{\mathrm{Vol}^d(IR(p_1, q_2))} \tag{12}$$

where $IR(p_1, q_1)$ and $IR(p_1, q_2)$ are two consecutive independent regions, $\mathrm{Vol}^d(IR(p_1, q_2))$ denotes the volume of $IR(p_1, q_2)$ in a $d$-dimensional space and $\mathrm{Vol}^d(IR(p_1, q_1)) \geq \mathrm{Vol}^d(IR(p_1, q_2))$.

Moreover, the volume of the overlapping region of two independent regions (two spheres) can be calculated as follows (see Figs. 18, 19).

$$\begin{aligned}&\mathrm{Vol}^d(IR(p_1, q_1) \cap IR(p_1, q_2)) \\ &= \int_{u_0}^{r_1} \mathrm{Vol}^{d-1}(h)\mathrm{d}u + \int_{t_0}^{r_2} \mathrm{Vol}^{d-1}(h)\mathrm{d}t\end{aligned} \tag{13}$$

where $\mathrm{Vol}^{d-1}(h)$ denotes the volume of the sphere with radius $h$ in a *(d-1)*-dimensional space, $h = (r_1^2 - u^2)^{1/2} = (r_2^2 - t^2)^{1/2}$. $u_0$ and $t_0$ are the lower bounds of the integrals, where $u_0 = \frac{r_1^2 - r_2^2 + D(q_1,q_2)^2}{2D(q_1,q_2)}$ and $t_0 = \frac{r_2^2 - r_1^2 + D(q_1,q_2)^2}{2D(q_1,q_2)}$. $D(q_1, q_2)$ denotes the distance between $q_1$ and $q_2$.

Figure 18 shows an example of the independent region merging in a 2-dimensional space. Line $l_{pp'}$ decomposes the overlapping region into two sub-regions. The length of $l_{pp'}$ is denoted by $V^1(h) = 2h$. If we move $l_{pp'}$ toward $q_1$ and $q_2$, respectively, then, the volume of the two sub-regions is the sum of integral of $V^{d-1}(h)$ in the overlapping area. In a $d$-dimensional space, $l_{pp'}$ becomes a sphere in a *(d-1)*-dimensional hyper-plane, and $h$ is the radius of the sphere.

In a 2-dimensional space, $ratio(q_1, q_2)$ can be calculated as follows,

$$
\begin{aligned}
&ratio(q_1, q_2)\\
&= \frac{\mathrm{Vol}^2(IR(p, q_1)) \cap \mathrm{Vol}^2(IR(p, q_2))}{\mathrm{Vol}^2(IR(p, q_1))}\\
&= \frac{\int_{u_0}^{r_1}(h)\mathrm{d}u + \int_{t_0}^{r_2}(h)\mathrm{d}t}{\mathrm{Vol}^2(IR(p, q_1))}\\
&\approx \frac{r_1^2 \cos^{-1}\left(\frac{d^2+r_1^2-r_2^2}{2dr_1}\right) + r_2^2 \cos^{-1}\left(\frac{d^2+r_2^2-r_1^2}{2dr_2}\right)}{\pi {r_1}^2}
\end{aligned}
\tag{14}
$$

#### 5.5.3 Duplicate elimination

The third issue is that our solution may produce duplicates since a data point may locate in two or more independent regions. If the data point is a spatial skyline, it will be written to the results of the query by multiple reducers. To eliminate the duplicates, we associate a unique independent region identifier to each data point, which indicates that the data point will be output as a spatial skyline by the reducer which processes data points in the independent region. Reducers processing data points in other independent regions will not output the data point even if it is a spatial skyline. Take Fig. 11 as an example, $p_5$ is a data point in $IR(p_1, q_2)$ and $IR(p_1, q_3)$. If the identifier of $IR(p_1, q_2)$ is associated with $p_5$, and $p_5$ is a spatial skyline, $p_5$ is output only by the reducer processing data points in $IR(p_1, q_2)$.

## 6 Experimental validation

As we mentioned in Sect. 2.2, data have to be copied into GPU memory to be managed in GPU. GPU memory size is relatively limited compared to CPU memory. The input data will not fit in GPU memory if the data size is too large. Due to the limitation of GPU memory, we only conduct the experiments on up to 10 million data points. In real world, input dataset could achieve 10 million or even 500 million. The GPU solution is no longer suitable for large input datasets. We propose a MapReduce-based parallel solution for large input datasets. We test the solution on 100 to 500 million data points.

***Datasets.*** To test on small datasets, we use real-world datasets downloaded from Geonames[1] and set 10 million data size by default. To test on large datasets, we use large size synthetic datasets that are randomly generated under uniform distribution in a 2-dimensional space and set 100 million data size by default. Similar to [9], the query points are randomly generated in random regions where maximum MBR(Q) is 1% of the entire dataset. We use 10 query points by default.

***Algorithms.*** We introduce the algorithms for MapReduce and GPU settings.

*MapReduce setting.* Our proposed algorithm is denoted by $PSSKY\text{-}G\text{-}IR\text{-}PR$, which combines the concepts of independent region, pruning region and multi-level grid data structure for efficient query evaluation.

We developed two single-phase MapReduce-based solutions as baselines, $PSSKY$ and $PSSKY\text{-}G$. $PSSKY$ applies a random data partitioning method to split data points.

Each mapper uses BNL to produce local spatial skylines by comparing every pair of data points, and a reducer merges the local results and outputs the global spatial skylines. $PSSKY\text{-}G$ works similarly to $PSSKY$ except that $PSSKY\text{-}G$ utilizes multi-level grid data structure for efficient spatial dominance tests. Since all three solutions use the same algorithm in convex hull computation, we will focus primarily on the investigation of the overall performance of solutions and the effect of independent regions and pruning regions on spatial skyline computation in the second and third MapReduce phases. All solutions were implemented in Java on Hadoop 2.6, which is an open source implementation of the MapReduce framework [26].

*GPU setting.* We downloaded the simulation codes of BSkyTree [14] and multi-core *Hybrid* [27], which are state-of-the-art parallel skyline algorithms. We convert spatial skyline queries into general skyline queries by adding an additional phase at the beginning of the two methods to calculate distance between every pair of data point and query point by using the GPU in parallel. After the distance calculation, the two methods can work as baseline algorithms in our experiments.

We also reproduced SkyAlign [22] as a GPU-based baseline algorithm in Java and JCuda 7.0, denoted by *Spatial-GPU*. Because our *Spatial-GPU* is implemented in Java, we use the same setting to guarantee fair comparison. In terms of run-time evaluating for $SkyAlign$, we omit repacking and sorting consumption and only take GPU-related computation into consideration. In this way, we make sure our reproduction of repacking or sorting will not affect the execution time evaluation.

The code of our implementation is publicly available[2].

***Configuration.*** We introduce the configuration for MapReduce and GPU settings.

*MapReduce setting.* The experiments were conducted on a 12-node shared-nothing cluster. Each node is equipped with 19 Intel Xeon 2.2 GHz processors and 128 GBytes of memory. All nodes were connected by GigaBit Ethernet network.

[1] http://www.geonames.org/.

[2] https://github.com/VV123/SpatialSkyline-GPU.

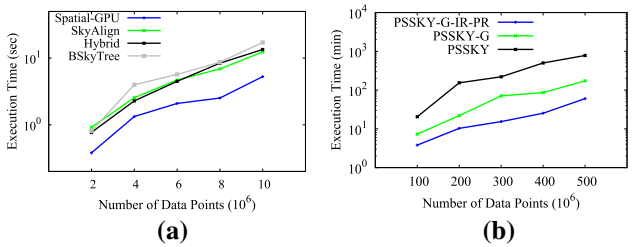


**Fig. 20** Run-time performance varying dataset cardinality. **a** Small-scale data. **b** Large-scale data

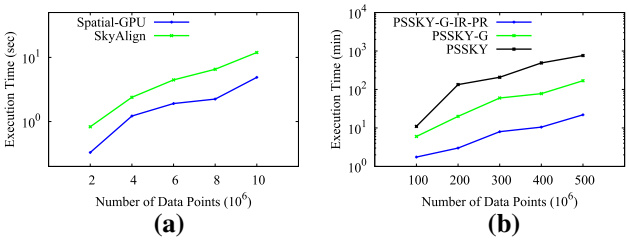


**Fig. 21** Spatial skyline execution time varying dataset cardinality. **a** Small-scale data. **b** Large-scale data

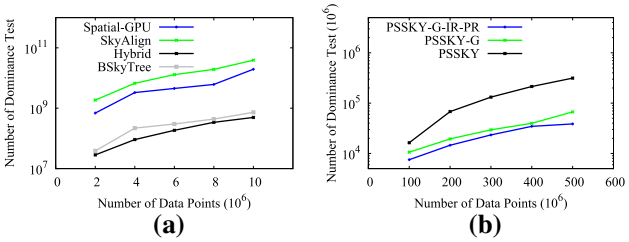


**Fig. 22** Number of dominance test varying dataset cardinality. **a** Small-scale data. **b** Large-scale data

All results were recorded after the system model reached a steady state.

*GPU setting*. The GPU algorithms use an Nvidia Tesla K80 graphics card. K80 has 12 GB GDDR5 memory per GPU, which is the maximum memory size on the market at the time of this writing. The graphics card is equipped with Intel(R) Xeon(R) E5-2670 v3 2.30 GHz CPU processor and 256 GBytes of CPU memory. We assume all data have been loaded into CPU memory; the time of loading the data to CPU memory is excluded from our evaluation. The GPU implementations are compiled using nvcc 7.5 compiler. *Hybrid* runs with 8 threads following [22].

## 6.1 Scalability with cardinality

First of all, we evaluate the effect of data cardinality on all solutions.

We apply small-scale real-world dataset on GPU algorithms, and large-scale synthetic dataset on MapReduce setting. Query points are set to 10 by default.

As shown in Fig. 20, we vary the cardinality from 2 to 10 million, and 100 to 500 million, respectively. All algorithms consume more time as cardinality grows, and our algorithms perform well constantly. The execution time and number of dominance tests are not correlated for GPU setting (shown in Figs. 20, 22), but are correlated for MapReduce setting. In Fig. 21, we measure the pure spatial skyline time, which refers to GPU kernel time in GPU setting, and reducer phase at the final computation phase in MapReduce algorithms.

### 6.1.1 GPU algorithms scalability with cardinality

First of all, we evaluate the effect of data cardinality on small-scale real-world dataset. As Fig. 20a shows, the execution time of all solutions increases when dataset grows. However, the growth rate of *Spatial-GPU* is significantly lower than *BSkyTree*, *Hybrid* and *SkyAlign*. *Spatial-GPU* executes more than 40% faster than CPU algorithms. Spatial skyline is high-dimensional because the number of query points is relatively large ([22] tests up to 16 dimensions). *SkyAlign* consists of $d$ sequential iterations ($d$ = the number of dimensions), which is not able to fully take advantage of GPU parallel features.

Our real-world dataset is more likely to be a non-uniform dataset, and our result is consistent with results shown in [22] that CPU algorithms work well for correlated dataset. As we observed in [22], *Hybrid* outperforms *BSkyTree* for anti-correlated and independent datasets and in contrast for correlated dataset. As shown in Fig. 20a, the results are consistent with the claim in [22] that *SkyAlign* does not perform better than CPU algorithms on low cardinality correlated datasets.

*SkyAlign* uses recursively partitioned algorithms and does not require more dominance tests when the number of points increases because the resulting quad-tree becomes deeper [9]. However, *SkyAlign* still suffers from sequential iterations, which could not be optimized by GPU parallel scheme. Moreover, *SkyAlign* repacks data points and $MT$ and $QT$ arrays for every iteration.

For low-dimensional data or easier workloads, GPU algorithms are ineffective because the computational task is not challenging enough to amortize the cost of retrieving the data to the GPU memory. The result is also mentioned in [22] that CPU algorithms are efficient enough for low-dimensional data.

We also test the number of dominance tests in Fig. 22a. Even though all GPU algorithms suffer from more dominance tests than CPU algorithms, the execution time is still better. The reason is that GPU executes dominance tests in parallel. Compared within CPU algorithms, the number of dominance tests and run-time are correlated. For GPU algo-

rithms, run-time does not fully depend on the total number of dominance tests.

In terms of GPU algorithms, our theory is that *SkyAlign* suffers from sequential iterations among dimensions, and *Spatial-GPU* filters in parallel , which makes the most of the GPU paradigm. In summary, GPU algorithms are suitable for high-dimensional relatively large cardinality datasets.

#### 6.1.2 MapReduce algorithms scalability with cardinality

As Fig. 20b displays, the execution time of all solutions increases when the datasets grow, which is consistent with the number of dominance tests as shown in Fig. 22b. The growth rate of $PSSKY\text{-}G\text{-}IR\text{-}PR$ is lower than that of $PSSKY$ and $PSSKY\text{-}G$. In addition, on average, $PSSKY\text{-}G\text{-}IR\text{-}PR$ executes around 90% faster than $PSSKY$ and 32% faster than $PSSKY\text{-}G$, respectively. The reason is that $PSSKY\text{-}G\text{-}IR\text{-}PR$ is able to parallelize the spatial skyline evaluation by applying the concept of independent region and efficiently filter out unqualified data points in pruning regions. Moreover, a performance improvement was observed when comparing $PSSKY\text{-}G$ with $PSSKY$ because the multi-level grid data structure is employed to efficiently access the proximity information of data points for the dominance test.

In summary, no matter for GPU-based algorithms or MapReduce-based algorithms, all methods exhibit the basic trend of increasing run-time with respect to the increases in cardinality, and our methods achieve high efficiency with different dataset cardinalities.

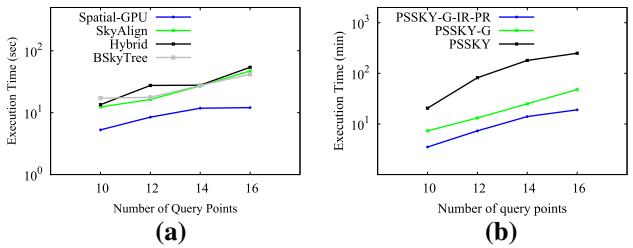


**Fig. 23** Run-time varying number of query points. **a** Small-scale dataset. **b** Large-scale dataset

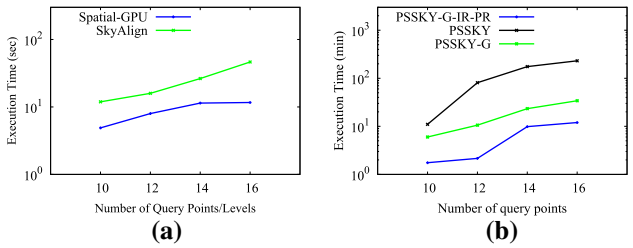


**Fig. 24** Spatial skyline execution time varying number of query points. **a** Small-scale dataset. **b** Large-scale dataset

### 6.2 Effect of query points

We investigate the effect of query points on different data scales. We fix the size of data points at 10 and 100 million, respectively. The number of convex hull query points are 10, 12, 14 and 16. In this section, query point means convex hull query point. Figure 23 displays the overall execution time for small-scale and large-scale datasets, which justifies our theory that our system maintains high efficiency in different settings. Figure 24 displays the pure spatial skyline execution time which is kernel time in GPU setting and reduce phase in Hadoop setting. Figure 25 presents the number of dominance tests. Figures 23 and 24 show that pure spatial skyline time and overall execution time share the same trend, which means spatial skyline operations are the major workload. However, the number of dominance tests is not fully related to execution time for GPU setting, which is consistent with our theory that increasing parallelization is able to improve efficiency dramatically.

#### 6.2.1 Effect of query points for GPU algorithms

We fix the size of data points at 10 million for GPU setting. Figure 23a displays the overall execution time of four solutions. With the increasing number of query points, the execution time of all four algorithms increases. GPU algorithms do not exhibit much benefit for fewer query points. Because we adopt the IRG-based parallel filter, more query points will not increase the filter time. When a spatial skyline query has more query points, the dominance test for each pair of data points requires more comparisons, which will make better use of GPU parallel scheme.

For CPU algorithms, no matter how the number of query points changes, our *Spatial-GPU* always exhibits faster execution time.

Figure 24a shows the kernel time corresponding to Fig. 23a. Both figures share the same trend, but the GPU algorithm suffers from higher overhead due to data transfer and takes more execution time proportionally.

#### 6.2.2 Effect of query points for MapReduce algorithms

We fix the size of data points at 100 million. The number of query points selected are 10, 12, 14 and 16. Figure 23b displays the overall execution time. The experimental results show that the entire process of query evaluation takes longer with increasing number of query points. The reason is that there are more data points in the search region, and the number of data points requiring dominance test becomes larger. In Fig. 2 for example, a convex hull is represented by $q_1$, $q_2$ and $q_3$. $p_1$ is used to generate three independent regions. If more query points form a new convex hull while existing independent regions stay the same, we can conclude that new

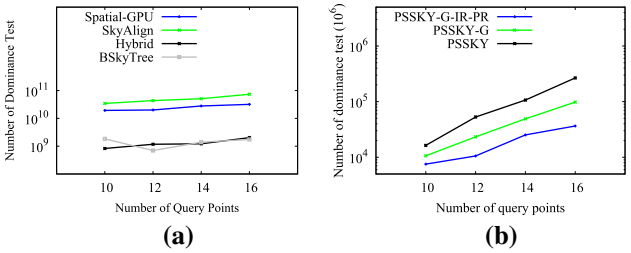


**Fig. 25** Number of dominance test varying number of query points. **a** Small-scale dataset. **b** Large-scale dataset

query points will form more independent regions. The overall area covered by all the independent regions will increase accordingly. Thus, more data will be located in the independent regions and be processed by reducers in the third MapReduce phase.

Figure 24b shows the execution time of spatial skyline computation, and the number of dominance tests grows rapidly when the number of query points grows larger. Similar results are observed in terms of the number of dominance tests in Fig. 25b.

## 6.3 GPU multi-threading scheme experimental validation

In this section, we evaluate the unique features of GPU multi-threading scheme design. First, we investigate the thread divergence of *SkyAlign*. Then, we investigate the effect of the number of leaf nodes for *Spatial-GPU*. We also investigate the effect of pre-filter levels in *Spatial-GPU*. Finally, we test the effect of GPU hardware.

### 6.3.1 Latency hiding

As we introduced in Sect. 2.2, the efficiency of GPUs heavily depends on latency hiding. During every GPU cycle, the warp scheduler selects an idle warp from the warp pool for execution. If there is any instruction ready in the selected buffer of warp instructions without pipeline hazard, such warp instruction is ready for execution; unless it is stalled for the following factors: cache miss, data hazard, control hazard, structural hazard, or delays caused from the warp scheduling policy.

The latency-hiding ability of GPUs has been well investigated in [28]. They draw the conclusion that the fast context switching and massive multi-threaded architecture can effectively hide most latency by swapped in and out warps. Their conclusion is consistent with our experiments. Figure 26 shows the number of dominance tests in *SkyAlign* in each iteration. In this example, 10 query points are used, which means after transferring to general skyline, there will be 10 dimensions and 10 iterations in *SkyAlign* as well.

In Fig. 26, the gray line shows the *maximum* number of dominance tests in a thread, and the black line shows the *average* number of dominance tests in a thread. The $x$-axis represents the number of iterations. All the points are arranged in log scale. We can observe a large variance among each iteration and intense fluctuation along iterations. In other words, the workload of each thread is uncertain and there is a high chance that threads in a warp diverge dramatically.

On the contrary, *Spatial-GPU* makes the most of the GPU parallel mechanism and provides fast context switch and massive threads. We use multi-level independent region-based pre-filter method to filter out the majority of data in parallel. Since dominance tests are unavoidable for the rest of the data, we simply apply one dominance test per thread or one point per thread. *SkyAlign* computes $10^2$ to $10^3$ of dominance test each thread. When using Parallel Dominance Test, *Spatial-GPU* deploys $10^8$ threads and computes one dominance test each thread.

In summary, there are two reasons contributing to the overall execution time difference between *SkyAlign* and *Spatial-GPU*. First, branch divergence, small size of threads and heavy workload per thread limit the performance of latency hiding. Second, the advanced multi-level pre-filter method dramatically decreases the number of spatial skyline candidates in parallel and decreases the number of dominance tests in quadratic.

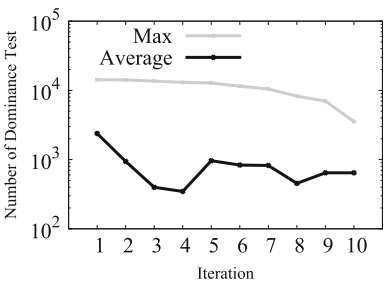


**Fig. 26** *SkyAlign* dominance tests per iteration

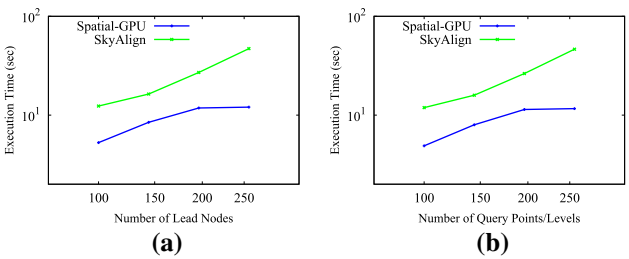


**Fig. 27** Varying number of leaf nodes of $L^{|Q|+1}$IRG. **a** Run-time performance. **b** Kernel time

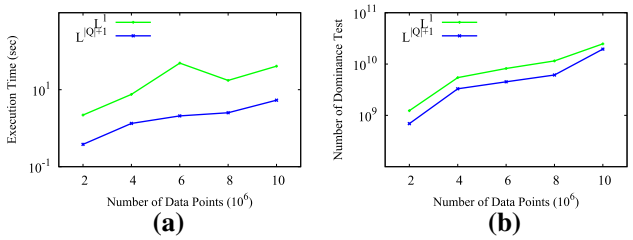


**Fig. 28** Using levels for multi-level IRGs. **a** Execution time. **b** Dominance tests

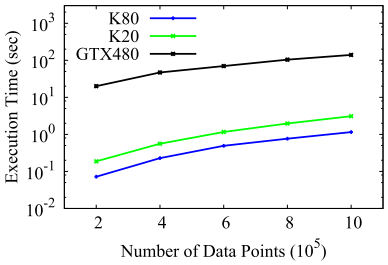


**Fig. 29** The overall execution time of varying GPU hardware

#### 6.3.2 Effect of leaf nodes in MIRGs

As shown in Fig. 27, we test the run-time performance and kernel time for all GPU-based algorithms varying number of leaf nodes in MIRGs. We use level $|Q| + 1$ for query size $\{10, 12, 14, 16\}$, and leaf nodes are approximately $\{100, 150, 200, 250\}$.

From the experimental results, we can observe that *Spatial-GPU* minimizes the execution time due to multi-level IRG-based parallelization. With increasing number of leaf nodes, the run-time for *Spatial-GPU* is relatively stable. All methods exhibit the same basic trend of increasing running times with respect to increases in the number of leaf nodes.

In theory, *Spatial-GPU* filters in parallel and the overall execution time should not increase with varying numbers of leaf nodes. However, the run-time increases with more leaf nodes from the experimental results. In practice, the computational task within each independent region can be executed in parallel, but there is a reduction phase that merges intermediate results from all regions together. Moreover, data points for each independent region cannot fit into the shared memory and have to be stored in GPU memory. In this way, all the independent regions read from GPU memory at the same time, and they are not fully "parallelized" in practice.

#### 6.3.3 Multi-level pre-filter comparison

We also test how pre-filter levels affect the performance of *Spatial-GPU* in Fig. 28. Figure 28a, b shares exactly the same trend. As we expected, the $L^1$ pre-filter method suffers from longer execution time and more dominance tests on average compared with the $L^{|Q|+1}$ pre-filter method. This observation justifies that the $L^{|Q|+1}$ pre-filter method not only amortizes the overhead for implementing it, but also reduces a large portion of spatial skyline comparisons.

#### 6.3.4 Performance of difference GPUs

We test our *Spatial-GPU* algorithm on different GPUs
*GeForce GTX480* Each GPU has 1.5GB GDDR5 memory, 480 CUDA cores and memory bandwidth 177.4 GB/sec. Arithmetic rate is 1.35 Tflops for single precision and 168 Gflops for double precision.
*NVIDIA Tesla K20* Each GPU has 5GB GDDR5 memory, 2496 CUDA cores and memory bandwidth up to 208GB /sec. Arithmetic rate is 3.52 Tflops for single precision and 1.17 Tflops for double precision.
*NVIDIA Tesla K80* Each GPU has 12GB GDDR5 memory, 2496 CUDA cores and memory bandwidth of 480GB/sec. Arithmetic rate is 8.74 Tflops for single precision and 2.91 Tflops for double precision.

The performance of GTX480, K20, K80 are in ascending order, same as the GPU memory space. For GTX480, the GDDR5 memory is only 1.5GB, so we test the dataset of $\{2, 4, 6, 8, 10\} * 10^5$.

As shown in Fig. 29, the hardware performance will affect the execution time dramatically.

### 6.4 Hadoop MapReduce scheme experimental validation

In this section, we evaluate the unique features of Hadoop MapReduce scheme design. First of all, we investigate the effect of independent regions and pruning regions.

#### 6.4.1 Effect of independent regions and pruning regions on spatial skyline algorithms

To evaluate the effectiveness of independent regions and pruning regions, we compare the execution time of spatial skyline computation in $PSSKY\text{-}G\text{-}IR\text{-}PR$ (the execution time of reducers in the third MapReduce phase) with the ones in $PSSKY$ and $PSSKY\text{-}G$. The cardinality of datasets varies from 100 to 500 million points. As Fig. 21b shows, the execution time of all solutions increases when the datasets grow. The execution time of $PSSKY$ increases rapidly due to high complexity of spatial skyline computation. The growth rate of $PSSKY\text{-}G\text{-}IR\text{-}PR$ is the lowest because all data points can be processed in parallel and a significant portion of data points can be discarded without dominance test. Moreover, the reducer that merges spatial skylines becomes a bottleneck in $PSSKY$ and $PSSKY\text{-}G$, which consumes 50% to 90% of the total execution time.

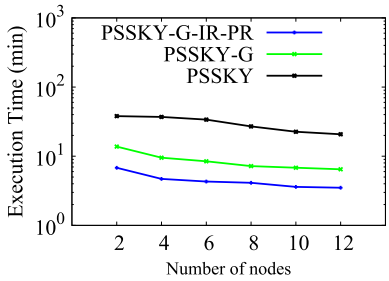


**Fig. 30** The overall execution time of the three solutions by varying node cardinality

#### 6.4.2 Effect of number of nodes

We evaluate the speedup of proposed solutions by scaling up the size of MapReduce cluster. The datasets are fixed at 100 million objects. The cardinality of cluster nodes varies from 2 to 12.

In Fig. 30, the execution time of all solutions drops as the size of the cluster increases. As expected, $PSSKY$ always consumes more execution time than $PSSKY\text{-}G$ and $PSSKY\text{-}G\text{-}IR\text{-}PR$ while scaling up the cluster. On average, $PSSKY\text{-}G\text{-}IR\text{-}PR$ enjoys the highest dropping rate. The dropping rate of $PSSKY$ is constantly lower than 20%. The reason is that more map or reduce tasks can be executed in parallel with more computing resources. However, although all three methods take advantage of mapper parallelism, only reducers of $PSSKY\text{-}G\text{-}IR\text{-}PR$ run in parallel because the global region is partitioned into independent regions. The skyline results in each independent region do not depend on the ones in other independent regions.

## 7 Related work

In this section, we review previous work related to spatial skyline queries, parallel solutions for general skyline queries and GPU solutions for general skyline queries.

### 7.1 Spatial skyline queries

As a special case of dynamic skyline queries, Spatial Skyline Queries (SSQ) can be addressed by Block Nested Loop (BNL) [1] and Branch-and-Bound Skyline algorithms (BBS) [4]. In a dynamic skyline query, each object is mapped to another search space by using pre-defined functions. All the objects that are not dominated by other objects in the search space after the mapping are returned from the dynamic skyline queries. BNL algorithm can address the dynamic skyline queries because it compares every pair of objects in the input dataset, and eliminates the ones that are dominated by any other objects. BNL does not need indices and is efficient over small datasets; however, it suffers from I/O access when the input datasets become large. If the size of skyline candidates exceeds the size of available memory space, all the candidates have to be written to a temporary data stream and read back when they are needed in the next iteration of object comparison. BBS relies on an R-tree to evaluate the general skyline queries; it calculates the $mindist$ of intermediate entries in the R-tree and searches the space by expanding the entry with the smallest $mindist$. Nevertheless, BBS does not consider the relation between the input query points and data points.

Motivated by the inefficiency of BNL and BBS, a Branch-and-Bound Spatial Skyline ($B^2S^2$) algorithm and a Voronoi-based Spatial Skyline ($VS^2$) algorithm were proposed for spatial skyline evaluation [9]. In addition to considering the properties of the convex hull generated by input query points, $B^2S^2$ searches the space by visiting an R-tree from top to bottom. Once the first spatial skyline is found, $B^2S^2$ expands the R-tree with the node which has the minimum *mindist* value and checks the dominance between the visited node and all spatial skyline candidates found so far. The process continues until all intermediate nodes potentially containing spatial skylines have been visited. On the other hand, $VS^2$ builds a Voronoi diagram over input data points. The input data points are organized by their Hilbert values in pages in order to preserve their locality. After completion of convex hull calculation, $VS^2$ starts with the closest data points to the query points and searches the space by visiting the neighbors of visited data points over the Voronoi diagram. For every visited data point, $VS^2$ compares it with all spatial skylines found so far for spatial dominance test. The process continues until all Voronoi cells (or data points) that potentially contain spatial skylines have been visited. Inspired by high cost of the spatial dominance test, $VS^2$ was improved by reducing the number of spatial dominance tests [10]. In addition to applying sorting techniques, the method is able to identify seed skyline points (a subset of spatial skyline points) without dominance test. Given a set of query points $Q$ and a set of data points $P$, let $V(p_i)$ be the Voronoi cell of data point $p_i \in P$, the seed skyline points are the points $p_i$ that $V(p_i)$ intersect with the boundary of the convex hull of $Q$ or are inside the convex hull. However, none of the aforementioned methods can address the spatial skyline query in parallel. $B^2S^2$ requires a pre-structured R-tree and $VS^2$ needs to build a Voronoi diagram over input data points. Extending their methods to a distributed and/or parallel environment is non-trivial.

### 7.2 Parallel skyline solution

Due to high cost of skyline evaluation, a number of advanced solutions have been proposed to evaluate the general skyline queries in a distributed and/or parallel environment. Balke et al. developed a parallel skyline solution over distributed

environments [29]. Their method first vertically partitions input datasets in such a manner that each partition keeps object attributes in one dimension. Then, the skyline objects are calculated in parallel and reported to a central point for a final dominance check. Wu et al. designed a parallel skyline method that leverages content-based data partitioning [30]. Their method can avoid unnecessary data access and can progressively produce skylines by using recursive region partitioning and dynamic region encoding mechanisms. Moreover, the incremental scalability is also provided in such a manner that workload can be automatically balanced by distributing objects to new nodes. In addition to random data partitioning methods that can generate similar data distribution in each partition [31] and grid-based data partitioning methods that consider object proximity [32,33], Vlachou et al. proposed an angle-based data partitioning method that partitions objects by their angular coordinates [34]. The average pruning power of objects within a partition can be increased and the number of skyline objects in local skyline calculation can be minimized by applying the angle-based partitioning method. Köhler et al. designed a hyper-plane based data partitioning method in order to minimize the local skylines in a partition and achieve efficient local skyline merging [35]. Moreover, a variety of MapReduce-based parallel solutions have been proposed for skyline queries and other database applications. Han et al. proposed an advanced skyline algorithm that utilizes Skyline with Sorted Positional index Lists (SSPL) to reduce I/O cost [36]. Zhang et al. implemented BNL, SFS and Bitmap algorithms using MapReduce framework [37]. Chen et al. applied an angular data partition in their MapReduce-based solution for skyline query evaluation [38]. Eldawy et al. developed CG_Hadoop, a suite of MapReduce algorithms, to solve fundamental computational geometry problems, which include convex hull computation [24]. Mullesgaard et al. investigated the general skyline queries by using the MapReduce framework. Their method uses bit strings to represent the dominance relation of attributes and generates independent partition groups for calculating local skyline objects in parallel [15]. Zhang et al. proposed an efficient parallel skyline solution using MapReduce, in which a Partial-presort Grid-based Partition Skyline (PGPS) algorithm was developed to significantly improve the merging skyline computation on large datasets [17]. More importantly, PGPS can be easily incorporated in the shuffle phase of the MapReduce framework with minor overhead. However, our proposed solution targets on spatial skyline queries, which are more challenging than the general skyline queries as each dominance check requires computing dynamic distance attributes. None existing computation algorithms could address the spatial skyline problems directly without extra overhead. Existing partition schemes did not take advantage of spatial properties thus could not fully accelerate spatial skyline problems. Therefore, we propose a novel partition method and a parallel algorithm which includes independent regions to parallelize the spatial skyline computation and pruning regions to reduce the cost of spatial dominance test.

A Distributed Spatial Skyline (DSS) algorithm [39] was proposed to collaboratively find the spatial skylines in wireless sensor networks. DSS parallelizes the search for skylines by partitioning the search space based on Voronoi cells. A geometry-based distributed spatial query strategy (GDSSky) [40] was also developed to ensure the efficient use of sensor energy. They use a regional partitioning strategy based on the triangulation method to reduce the WSN energy consumption. It also uses the Voronoi diagram method to divide the spatial region. Their scope is to reduce energy rather to support large-scale datasets, and generalizing Voronoi diagram to large-scale datasets is problematic. In general, DSS task is to find spatial skyline in wireless sensor networks that contain hundreds of nodes, which is different from our scope.

### 7.3 GPU-based skyline solutions

There are several GPU-based skyline solutions observed in literature. In the GNL algorithm, every object is assigned with a global counter (initialized with 0) [12]. Then, GNL starts a thread for every object: each thread compares the object with all objects in the candidate window (the window size is set to be half of input dataset size by default). If the object is dominated by any other object, its global counter will be increased by 1. After all dominance tests are completed, the objects with global counter equal to 0 are skylines. GNL is able to achieve high throughput since all dominance tests are operated in parallel.

The GGS algorithm follows the idea of Sort-First Skyline method (SFS) [5] and sorts the data by Manhattan Norm first [13]. Since objects cannot dominate any one that has been scanned before, GGS progressively outputs skylines and eliminates unqualified skyline candidates through iterations. In each iteration, GGS creates a candidate buffer with the first $\alpha$ objects and initiates threads to compare objects with the ones in candidate buffer. After conducting dominance tests, objects that are dominated are discarded, and objects remaining in the candidate buffer are produced as skylines. The process continues until all objects are compared. Although the dominance tests in GNL and GGS can be processed in parallel, the number of object comparisons may become unacceptably large. The computational complexity of both solutions may degrade to quadratic over large-scale dataset.

A recursive point-based partitioning method was proposed for GPU-based parallel solutions in [6,14]. Initially, a pivot point is selected, and the search space is partitioned to several sub-spaces by the pivot point. Then, pivot points are recur-

sively selected in every sub-space with all the pivot points organized in a tree. In the process of pivot selection, every object will be compared with pivot points encountered in the path of traversing the tree top to down. If an object is not dominated by any other object, it will be the next pivot point, then added to the tree as a leaf node. The pivot selection can be performed in parallel; however, the pivot tree construction and updating become a bottleneck since the global tree is shared by all the processes. Moreover, the pivot-based partitioning method is sensitive to the pivot selection. The hybrid multi-core algorithm is a point-based method that flattens the tree to an array for faster memory accessing patterns, it also processes data points within blocks of size $\alpha$ to improve parallelization.

A parallel skyline algorithm SkyAlign [22], which utilizes a global static partitioning scheme was proposed for GPU computing scheme. To reduce the number of object comparisons in tree traversals, data-ordering and conditional branching, SkyAlign uses *controlled branching* to explore transitive relationships and avoid object comparisons. Initially, a GPU-friendly partitioning scheme is applied; three global virtual pivots are generated by the quartiles of the input dataset. Two bitmasks of each object are calculated by the relationship between the object and the virtual pivots. After initialization, every object is set to one of the $4^d$ partitions (sub-spaces). Then, SkyAlign sorts objects by grid cells (or partitions), and threads are mapped on the sorted layout. In a $d$-dimensional space, SkyAlign needs $d$ iterations during which remaining objects are compared with all the points using mask tests and dominance test if necessary. At the end of each iteration, dominated objects are discarded and the remaining objects are repacked for better locality.

Our proposed solution is different from point-based partitioning methods or SkyAlign. To maximize the parallelism and minimize the cost of object comparisons, our method applies an independent group-oriented partitioning scheme, in which objects are grouped by independent groups. A large number of object comparisons are avoided due to the pre-filter process. The spatial dominance of objects in an independent group does not rely on any objects outside that independent group. All spatial dominance tests in independent groups can be processed in parallel. There might be objects in two or more independent groups because independent groups may overlap. The duplicate skyline candidates can be removed with minimal overhead in our method.

## 8 Conclusion

In this paper, we propose two advanced parallel spatial skyline solutions utilizing GPU and MapReduce framework, separately. We demonstrate the efficiency and effectiveness of the proposed solutions through extensive experiments on different cardinality of real-world and synthetic datasets.

GPU multi-threading algorithm can solve spatial skyline queries efficiently. However, as data grow rapidly, addressing skyline queries on large-scale datasets in a single-node environment becomes impractical. Due to the memory limitation of GPU hardware, we apply MapReduce scheme for large-scale input datasets. The results show that MapReduce scheme is able to solve large-scale input datasets efficiently.

In the future, we plan to extend our solution from two perspectives. Firstly, we plan to extend the proposed parallel solution to address spatial skyline queries on road networks. Theoretically, the concepts of independent regions and pruning regions can be applied in the space of road networks. However, more investigation is needed to evaluate the cost of calculating the independent regions and pruning regions. Secondly, as the development of UAV drones continues, spatial skyline queries in 3-d space become practical. We also plan to implement and test our approach on higher-dimensional space.

**Acknowledgements** This research has been funded in part by the National Science Foundation grants IIS-1618669 (III) and ACI-1642133 (CICI).

## References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
2. Tan, K-L., Eng, P.-K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB, pp. 301–310 (2001)
3. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: VLDB, pp. 275–286 (2002)
4. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. **30**(1), 41–82 (2005)
5. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: ICDE, pp. 717–719 (2003)
6. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: SIGMOD Conference, pp. 483–494 (2009)
7. Sarma, A.D., Lall, A., Nanongkai, D., Xu, J.: Randomized multi-pass streaming skyline algorithms. PVLDB **2**(1), 85–96 (2009)
8. Huang, Z., Jensen, C.S., Lu, H., Ooi, B.C.: Skyline queries against mobile lightweight devices in MANETs. In: ICDE, p. 66 (2006)
9. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: VLDB, pp. 751–762 (2006)
10. Son, W., Lee, M.-W., Ahn, H.-K., Hwang, S.-W.: Spatial skyline queries: an efficient geometric algorithm. In: SSTD, pp. 247–264 (2009)
11. Hose, K., Vlachou, A.: A survey of skyline processing in highly distributed environments. VLDB J. **21**(3), 359–384 (2012)
12. Choi, W., Liu, L., Yu, B.: Multi-criteria decision making with skyline computation. In: Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference, pp. 316–323. IEEE (2012)
13. Bøgh, K.S.. Assent, I., Magnani, M.: Efficient GPU-based skyline computation. In: Proceedings of the Ninth International Workshop on Data Management on New Hardware, p. 5. ACM (2013)

14. Lee, J., Hwang, S.-W.: Scalable skyline computation using a balanced pivot selection technique. Inf. Syst. **39**, 1–21 (2014)
15. Mullesgaard, K., Pedersen, J.L., Lu, H., Zhou, Y.: Efficient skyline computation in MapReduce. In: EDBT (2014)
16. Park, Y., Min, J.-K., Shim, K.: Parallel computation of skyline and reverse skyline queries using mapreduce. PVLDB **6**(14), 2002–2013 (2013)
17. Zhang, J., Jiang, X., Ku, W.-S., Qin, X.: Efficient parallel skyline evaluation using MapReduce. IEEE Trans. Parallel Distrib. Syst. **27**(7), 1996–2009 (2016)
18. Zhang, C., Li, F., Jestes, J.: Efficient parallel kNN joins for large data in MapReduce. In: EDBT, pp. 38–49 (2012)
19. Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using MapReduce. In: SIGMOD Conference, pp. 495–506 (2010)
20. Okcan, A., Riedewald, M.: Processing theta-joins using MapReduce. In: SIGMOD Conference, pp. 949–960 (2011)
21. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
22. Bøgh, K.S., Chester, S., Assent, I.: Work-efficient parallel skyline computation for the GPU. Proc. VLDB Endow. **8**(9), 962–973 (2015)
23. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension. Discrete Comput. Geom. **10**(1), 377–409 (1993)
24. Eldawy, A., Li, Y., Mokbel, M.F., Janardan, R.: CG_Hadoop: Computational geometry in MapReduce. In: SIGSPATIAL, pp. 294–303 (2013)
25. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, New York (2008)
26. Apache, H.: http://hadoop.apache.org. Accessed 26 Apr 2016
27. Chester, S., Šidlauskas, D., Assent, I., Bøgh, K.S: Scalable parallelization of skyline computation for multi-core processors. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 1083–1094. IEEE (2015)
28. Lee, S.-Y., Wu, C.-J.: Characterizing the latency hiding ability of GPUS. In: Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium, pp. 145–146. IEEE (2014)
29. Balke, W.-T., Güntzer, U., Zheng, J. X.: Efficient distributed skylining for web information systems. In: EDBT, pp. 256–273 (2004)
30. Wu, P., Zhang, C., Feng, Y., Zhao, B.Y., Agrawal, D., El Abbadi, A.: Parallelizing skyline queries for scalable distribution. In: EDBT, pp. 112–130 (2006)
31. Cosgaya-Lozano, A., Rau-Chaplin, A., Zeh, N.: Parallel computation of skyline queries. In: HPCS, p. 12 (2007)
32. Afrati, F.N., Koutris, P., Suciu, D., Ullman, J.D.: Parallel skyline queries. In: ICDT, pp. 274–284 (2012)
33. Rocha-Junior, J.B., Vlachou, A., Doulkeridis, C., Nørvåg, K.: AGiDS: a grid-based strategy for distributed skyline query processing. In: Globe, pp. 12–23 (2009)
34. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Angle-based space partitioning for efficient parallel skyline computation. In: SIGMOD Conference, pp. 227–238 (2008)
35. Köhler, H., Yang, J., Zhou, X.: Efficient parallel skyline processing using hyperplane projections. In: SIGMOD Conference, pp. 85–96 (2011)
36. Han, X., Li, J., Yang, D., Wang, J.: Efficient skyline computation on big data. IEEE Trans. Knowl. Data Eng. **25**(11), 2521–2535 (2013)
37. Zhang, B., Zhou, S., Guan, J.: Adapting skyline computation to the Mapreduce framework: algorithms and experiments. In: DASFAA Workshops, pp. 403–414 (2011)
38. Chen, L., Hwang, K., Wu, J.: Mapreduce skyline query processing with a new angular partitioning approach. In: IPDPS Workshops, pp. 2262–2270 (2012)
39. Yoon, S., Shahabi, C.: Distributed spatial skyline query processing in wireless sensor networks. In: Proceedings of the IPSN, San Francisco, CA, USA, pp. 13–16 (2009)
40. Wang, Y., Song, B., Wang, J., Zhang, L., Wang, L.: Geometry-based distributed spatial skyline queries in wireless sensor networks. Sensors **16**(4), 454 (2016)