# Multidimensional Range Selection

Timothy M. Chan and Gelin Zhou(✉)

David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, Canada
{tmchan,g5zhou}@uwaterloo.ca

**Abstract.** We study the problem of supporting (orthogonal) *range selection* queries over a set of $n$ points in constant-dimensional space. Under the standard word-RAM model with word size $w = \Omega(\lg n)$, we present data structures that occupy $O(n \cdot (\lg n / \lg\lg n)^{d-1})$ words of space and support $d$-dimensional range selection queries using $O((\lg n / \lg\lg n)^d)$ query time. This improves the best known data structure by a factor of $\lg\lg n$ in query time. To develop our data structures, we generalize the "parallel counting" technique of Brodal, Gfeller, Jørgensen, and Sanders (2011) for one-dimensional range selection to higher dimensions.

As a byproduct, we design data structures to support $d$-dimensional range counting queries within $O(n \cdot (\lg n / \lg w + 1)^{d-2})$ words of space and $O((\lg n / \lg w + 1)^{d-1})$ query time, for any word size $w = \Omega(\lg n)$. This improves the best known result of JaJa, Mortensen, and Shi (2004) when $\lg w \gg \lg\lg n$.

## 1 Introduction

Range searching is an important topic in data structures and computational geometry. Recently, there has been growing interest in so-called "range aggregate queries", where instead of reporting or counting points inside a query range, we want to compute some aggregate function over the weights of the points inside the query range. In this paper, we study the version of the problem for multidimensional orthogonal ranges (axis-aligned boxes), where the aggregate function is the median, or more generally, the $k$-th smallest element.

More precisely, we can formulate the *d-dimensional (orthogonal) range selection* problem as follows, by viewing the weights as an extra dimension. The coordinates of each input point $p$ are represented as a $(d+1)$-tuple $(p_1, p_2, \ldots, p_d, p_{d+1})$. A query range is a $d$-dimensional rectangle $R = [a_1..b_1] \times [a_2..b_2] \times \cdots \times [a_d..b_d]$, and a range selection query asks for the point whose coordinate in the $(d+1)$-st dimension is the $k$-th smallest among all input points contained in $R \times (-\infty, \infty)$.

The underlying model of computation in this paper is the standard word-RAM model [4] with word size $w = \Omega(\lg n)$. Under this model, bitwise and arithmetic operations including multiplication can be performed over machine words in $O(1)$ time. Without loss of generality, coordinates of points are assumed


K. Elbassioni and K. Makino (Eds.): ISAAC 2015, LNCS 9472, pp. 83–92, 2015.
DOI: 10.1007/978-3-662-48971-0_8

to fit in *rank space* [5]. Coordinates can be replaced with their ranks in the point set, by increasing the query time by the cost of $O(1)$ predecessor searches.

The one-dimensional case of the range selection problem has been well studied [2,3,7,8]. Krizanc et al. [8] proposed the problem, and their structures required either super linear space or $O(n^\epsilon)$ query time for some constant $\epsilon > 0$. Brodal et al. [2] presented a linear space data structure with only $O(\lg n/\lg\lg n)$ query time, by a novel application of bit-level parallelism. As shown by Jørgensen and Larsen [7], Brodal et al.'s linear space structure achieved optimal worst-case query time for any data structure within $O(n \cdot \text{polylog}(n))$ bits of space. Jørgensen and Larsen [7] further designed an adaptive data structure for one-dimensional range selection queries, which occupied linear space and required only $O(\lg k/\lg\lg n + \lg\lg n)$ query time to select the $k$-th smallest element in the range. More recently, Chan and Wilkinson [3] reduced the query time to $O(\lg k/\lg\lg n + 1)$ using the same amount of space.[1] Shallow cutting [9] played a central role in designing these adaptive data structures.

For the case of higher dimensions, Brodal et al. [2] pointed out that a $d$-dimensional range selection query could be reduced to $O(\lg n)$ $d$-dimensional range counting queries. As shown by JaJa et al. [6], each $d$-dimensional range counting query requires $O((\lg n/\lg\lg n)^{d-1})$ query time. Thus the overall query time for a $d$-dimensional range selection query would be $O(\lg n \cdot (\lg n/\lg\lg n)^{d-1})$. To the best of our knowledge, this is the only known result for multidimensional range selection queries.

In this paper, we present data structures that support $d$-dimensional range selection queries using $O(n \cdot (\lg n/\lg\lg n)^{d-1})$ words of space and $O((\lg n/\lg\lg n)^d)$ query time, for any constant integer $d \geq 1$. This improves the straightforward solution by a factor of $\lg\lg n$ in query time. To develop our data structures, we generalize Brodal et al.'s "parallel counting" technique [2] into higher dimensions. In the search for the $k$-th smallest point, we keep solving subproblems of finding the first non-negative integer in an increasing array, where the length of an array is bounded above by $O(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. Instead of performing binary search on each of these subproblems, we examine all integers in the array from the highest bits in a parallel fashion, to speed up the search. These integers are not stored explicitly and have to be retrieved at query time, where the retrievals are either multidimensional range counting queries or multidimensional "parallel counting" queries.

Along the way, we also improve JaJa et al.'s work for range counting queries [6] with some novel bit manipulation tricks, which may be of independent interest. Our data structures support $d$-dimensional range counting queries within $O((\lg n/\lg w + 1)^{d-1})$ query time and $O(n \cdot (\lg n/\lg w + 1)^{d-2})$ words of space, for any word size $w = \Omega(\lg n)$. When $w$ is $\lg^{\omega(1)} n$, this improves JaJa et al.'s $O((\lg n/\lg\lg n)^{d-1})$ query and $O(n \cdot (\lg n/\lg\lg n)^{d-2})$ space bounds [6].

The rest of this paper is organized as follows. Section 2 contains preliminaries. Section 3 defines and solves a problem that abstracts the bottleneck of selection

[1] The conference version claimed $O(\lg k/\lg w + 1)$ query time but it would require non-standard word operations.

queries. In Sects. 4 and 5, we apply the "abstract" problem to range selection queries and present our data structures.

## 2 Preliminaries

Let $[a..b]$ denote the set of integers from $a$ to $b$. For point $p = (p_1, p_2, \ldots, p_{d+1})$ and each $i \in [1..d+1]$, $p_i$ is referred to as the $i$-th coordinate of $p$. For two points $p = (p_1, p_2, \ldots, p_{d+1})$ and $q = (q_1, q_2, \ldots, q_{d+1})$, $p$ is said to be *dominated* by $q$ if $p_i \leq q_i$ for each $i \in [1..d+1]$. Let $\sigma$ be a fixed parameter, which will be set to be either $\lceil \lg^{\epsilon} n \rceil$ or $\lceil w^{\epsilon} \rceil$ for constant $0 < \epsilon < 1/d$. A point $p = (p_1, p_2, \ldots, p_{d+1})$ is said to be of *type* $d'$ if $p_i \in [1..\sigma]$ for each $i \in [d'+1..d+1]$, i.e., the last $d - d' + 1$ coordinates fit in a narrow range $[1..\sigma]$. A set of $m$ points is said to be of *type* $d'$ if all these $m$ points are of type $d'$ and the $i$-th coordinates of points are in rank space for each $i \in [1..d']$, i.e., they are drawn from $[1..m]$ and pairwise different. The input point set is of type $d + 1$.

To exploit abilities of the word RAM, it is a standard technique to pack a short list of sufficiently small integers into a machine word. We divide a word into *subwords* of $m$ bits, each storing the *two's complement representation* of a signed integer that ranges from $-2^{m-1}$ to $2^{m-1} - 1$. With this representation, a set of operations can be performed in parallel to integers of the packed list in $O(1)$ time, provided that each of these integers in the input and the output fits in $m$ bits: One can add a constant integer to, subtract a constant integer from, or bit shift all signed integers of a packed list. One can also add or subtract corresponding integers of two packed lists. One can even find the first non-negative integer or the last negative one in a packed list, given that multiplications are permitted [4].

## 3 The "Abstract" Problem

Let $s$, $b$, and $t$ be parameters satisfying that $(s + b + 2)t < w$ and $b \ll s$. Intuitively, $s$ denotes the "section size", and $b$ denotes the number of "carry bits". The *$j$-th section* of an integer $x$ is defined to be $\lfloor x/2^{sj} \rfloor \mod 2^s$.

Let $A[1..t]$ be an increasing sequence of $w$-bit signed integers, with $A[0] < 0$. The goal of our abstract problem is to find the smallest index $i^* > 0$ so that $A[i^* - 1] < 0 \leq A[i^*]$. However, the value of each $A[i]$ is not given explicitly; rather, each $A[i]$ is *decomposed* into a sequence of signed integers $A_0[i], A_1[i], \ldots$, satisfying the properties that $|A_j[i]| < 2^{s+b}$ and $A[i] = \sum_{j \geq 0} A_j[i] \cdot 2^{sj}$. (Note that for $b = 0$, the decomposition corresponds to precisely the sections of an integer, but the parameter $b$ offers more flexibility, which will be needed in our applications later.) We can only access the sequence $A$ using the following *oracles*:

- Given $1 \leq i \leq t$, return $A[i]$;
- Given $j \geq 0$, return the concatenation of the binary representations of $(A_j[1], \ldots, A_j[t])$, stored in a single word in which the $i$-th subword is equal to $A_j[i]$. (Because $(s + b + 2)t < w$, the result fits in a word.)

We resist to solve this problem with binary search directly, which would require $O(\lg t)$ time. Instead, we examine in a parallel manner the $A_j[i]$'s in decreasing order of $j$. The following lemma shows how to achieve a query time that is adaptive to the values of the $A[i]$'s.

**Lemma 1.** *The "abstract" problem described in this section can be solved within* $O(1 + \frac{1}{s} \cdot \lg \frac{A[t]-A[1]}{A[i^*]-A[i^*-1]})$ *word operations and oracle calls.*

*Proof.* Given an index $p$, we define $B_p[i] = \sum_{j \geq p} A_j[i] \cdot 2^{s(j-p)}$ for $1 \leq i \leq t$. Note that $B_p[1], \ldots, B_p[t]$ may not be in increasing order. However, as shown below, $B_p[i] \cdot 2^{sp}$ provides an approximation of $A[i]$:

$$\begin{aligned} \left|A[i] - B_p[i] \cdot 2^{sp}\right| &\leq \sum_{0 \leq j < p} 2^{s+b} \cdot 2^{sj} = 2^{s+b} \cdot \frac{2^{sp}-1}{2^s-1} \\ &< 2^{s+b} \cdot \frac{2^{sp}}{2^{s-1}} = 2^{sp+b+1}. \end{aligned} \tag{1}$$

We maintain a range $[\ell + 1..r]$ that contains $i^*$, as well as the concatenation of the binary representations of $(B_p[\ell], \ldots, B_p[r])$. We ensure the invariant that for $\ell \leq i \leq r$, $|B_p[i]| \leq 2^{b+1}$ before each iteration. Thus $(B_p[\ell], \ldots, B_p[r])$ can be packed into a single word in which the $i$-th subword equals to $B_p[i]$ for $\ell \leq i \leq r$, and the remaining bits are 0.

At the beginning of the algorithm, we compute $A[1]$ and $A[t]$ with two oracle calls, and set the initial value of $p$ to be $p_0 = \lceil \frac{1}{s} \cdot \lg(A[t] - A[1]) \rceil$. We also set the initial value of $\ell = 1$ and $r = t$. Then for each $i$, $|A[i]| \leq A[t] - A[1] \leq 2^{sp_0}$. By Inequality 1,

$$|B_{p_0}[i] \cdot 2^{sp_0}| \leq |A[i] - B_{p_0}[i] \cdot 2^{sp_0}| + |A[i]| < 2^{sp_0+b+1} + 2^{sp_0}.$$

This implies that $|B_{p_0}[i]| \leq 2^{b+1}$. In addition, we observe that

$$B_{p_0}[i] = A_{p_0}[i] + A_{p_0+1}[i] \cdot 2^s + A_{p_0+2}[i] \cdot 2^{2s} + \cdots \equiv A_{p_0}[i] \pmod{2^s}.$$

We then have $B_{p_0}[i] + 2^{b+1} = (A_{p_0}[i] + 2^{b+1}) \mod 2^s$ since $|B_{p_0}[i]| \leq 2^{b+1}$ and $b \ll s$. This formula allows us to initialize $(B_{p_0}[1], ..., B_{p_0}[t])$ using one oracle call and $O(1)$ word operations.

In each iteration of the algorithm, we decrement the value of $p$ and compute $(B_p[\ell], \ldots, B_p[r])$ using the following equation:

$$(B_p[\ell], \ldots, B_p[r]) = 2^s \cdot (B_{p+1}[\ell], \ldots, B_{p+1}[r]) + (A_p[\ell], \ldots, A_p[r]).$$

The computation requires $O(1)$ word operations and one oracle call. Note that for $\ell \leq i \leq r$, $B_p[i]$ fits in a subword, because $|B_{p+1}[i]| \leq 2^{b+1}$ and $|B_p[i]| \leq 2^{s+b+1} + 2^{s+b} < 2^{s+b+2}$. We then find the largest index $\ell'$ in $[\ell..r]$ with $B_p[\ell'] \leq -2^{b+1}$, and the smallest index $r'$ in $[\ell..r]$ with $B_p[r'] \geq 2^{b+1}$. As described in Sect. 2, $\ell'$ and $r'$ can be determined using $O(1)$ word operations [4].

By Inequality 1, we have $A[\ell'] < 0$ and $A[r'] > 0$. Thus subranges $[\ell..\ell']$ and $[r' + 1..r]$ can be discarded, and we know that $i^*$ is contained in $[\ell' + 1..r']$.

We then evaluate $A[\ell' + 1]$ and $A[r' - 1]$ by two oracle calls. The algorithm terminates if one of the following conditions holds: $i^* = \ell' + 1$ is returned if $A[\ell' + 1] > 0$, or $i^* = r'$ is returned if $A[r' - 1] < 0$. Otherwise, we reset $\ell = \ell' + 1$ and $r = r' - 1$. Note that $|B_p[\ell' + 1]|, \ldots, |B_p[r' - 1]| \leq 2^{b+1}$, so the invariant is maintained, and we can continue on to the next iteration.

Now we analyze the running time of the algorithm. After each iteration before termination,

$$\begin{aligned} A[i^*] - A[i^* - 1] \leq A[r] - A[\ell] &< (B[r] - B[\ell]) \cdot 2^{sp} + 2 \cdot 2^{sp+b+1} \\ &\leq 2 \cdot 2^{b+1} \cdot 2^{sp} + 2 \cdot 2^{sp+b+1} = 2^{sp+b+3}. \end{aligned}$$

Thus, $p \geq \frac{1}{s} \cdot [\lg(A[i^*] - A[i^* - 1]) - O(b)]$. The algorithm requires $O(1 + \frac{1}{s} \cdot \lg \frac{A[t]-A[1]}{A[i^*]-A[i^*-1]})$ oracle calls. □

## 4 Range Selection

In this section, we apply the above "abstract" problem to range selection queries. We use $t = \lceil \lg^{\epsilon} n \rceil$, section size $s = \lceil (1/2) \cdot \lg^{1-\epsilon} n \rceil$, and $b = \Theta(\lg \lg n)$ for constant $0 < \epsilon < 1/d$. We build a range tree over the $(d+1)$-st coordinates of points with branching factor $t$. Thus, the height of the range tree is $O(\lg n / \lg \lg n)$. Each node $v$ in the range tree represents a range $[a_v..b_v]$ and the set $S(v)$ of points whose $(d+1)$-st coordinates are in $[a_v..b_v]$. The leaf nodes in the range tree each represent a single point.

To answer a given range selection query with query range $R$ and rank $k$, we repeatedly solve subqueries of the following form: given an internal node $v$ and its children $v_1, \ldots, v_t$ in the range tree, find the child $v_{i^*}$ so that the desired answer is contained in $S(v_{i^*})$. To connect these subqueries with the "abstract" problem, we set $A[i] = N[i] - k$, where $N[i]$ is the number of points that fall into $R \times [a_{v_1}..b_{v_i}]$. We set $A_j[i] = N_j[i] - k_j$, where $N_0[i], N_1[i], \ldots$ is a sequence to be specified later that decomposes $N[i]$, and $k_j$ is the $j$-th section of $k$.

To compute $N[i]$ and the $N_j[i]$'s, we define the following two kinds of queries over a point set $S$ of type $d$ with $\sigma = \lceil \lg^{\epsilon} n \rceil$, for which the support is summarized in Lemma 2. The proof of Lemma 2 is deferred to Sect. 5.

- *dominance counting* queries: given a query point $q = (q_1, q_2, \ldots, q_{d+1})$, return the number of points in $S$ that are dominated by $q$;
- *parallel counting* queries: given a query $(q_1, q_2, \ldots, q_d, j)$ for some $j \geq 0$, return the concatenation of $(C_j[1], \ldots, C_j[t])$, where, for $1 \leq i \leq t$, $C_0[i], C_1[i], \ldots$ is a sequence that decomposes $C[i]$, the answer to the dominance counting query $(q_1, q_2, \ldots, q_d, i)$.

**Lemma 2.** *For any constant $0 < \epsilon < 1/d$, a point set $S$ of size $m \leq n$ and type $d'$ with $\sigma = \lceil \lg^{\epsilon} n \rceil$ can be stored in $O(m \lg \lg n \cdot (\lg n / \lg \lg n)^{d'-1})$ bits of space, so that (a) dominance counting queries and (b) parallel counting queries can be answered in $O((\lg n / \lg \lg n)^{d'-1})$ query time.*

To facilitate the use of Lemma 2, we transform each $S(v)$ into a point set $D(v)$ of type $d$. For each point $p \in S(v)$, we replace the first $d$ coordinates of $p$ with their ranks in $S(v)$, and replace $p_{d+1}$ with the index of $v$'s child that represents a set containing $p$.

Given a $d$-dimensional query range $R$, we can express it as additions and subtractions of $2^d = O(1)$ $d$-dimensional dominance ranges. Let these ranges be $z_1, z_2, \ldots, z_{2^d}$. Computing $N[i]$ and $A[i]$, which is essentially a $(d+1)$-dimensional range counting query, can be reduced to dominance counting queries over $D(v)$ for ranges $z_1 \times [1..i], \ldots, z_{2^d} \times [1..i]$, and can be done by Lemma 2(a). Let $N[i, z_\ell]$ be the result for $z_\ell \times [1..i]$ for $1 \le \ell \le 2^d$. By Lemma 2(b) we can decompose $N[i, z_\ell]$ into a sequence $N_0[i, z_\ell], N_1[i, z_\ell], \ldots$ In addition, $(N_j[1, z_\ell], \ldots, N_j[t, z_\ell])$ can be computed for $j \ge 0$. We define $N_j[i]$ to be sum of $N_j[i, z_\ell]$ over all $z_\ell$. Then the sequence $N_0[i], N_1[i], \ldots$ decomposes $N[i]$ and the sequence $A_0[i], A_1[i], \ldots$ decomposes $A[i]$, after increasing the parameter $b$ by $\log(2^d) = O(1)$.

Now we can finally support range selection queries. Starting with the root node, we define and compute the oracles as described above. After determining $i^*$, the query algorithm recurses on $v_{i^*}$ after setting $k = k - A[i^* - 1]$. We repeatedly apply Lemma 1 until we reach a leaf node $v$, and $a_v = b_v$ is the answer. The query algorithm requires solving $O(\lg n / \lg\lg n)$ "abstract" problems. We sum the cost of Lemma 1 over these $O(\lg n / \lg\lg n)$ subproblems. Observe that the sum of the logarithms of ratios in Lemma 1 is actually telescoping. The total number of oracle calls is thus $O(\lg n / \lg\lg n + \frac{1}{s} \cdot \lg n) = O(\lg n / \lg\lg n)$, each requiring $O((\lg n / \lg\lg n)^{d-1})$ time. We conclude:

**Theorem 1.** *Under the word RAM model with word size $w = \Omega(\lg n)$, $d$-dimensional range selection queries over a set of $n$ points can be supported in $O((\lg n / \lg\lg n)^d)$ query time and $O(n \cdot (\lg n / \lg\lg n)^{d-1})$ words of space.*

## 5 Dominance Counting and Parallel Counting

Our method for dominance counting queries is similar to JaJa et al.'s work [6]. The major improvement is a novel algorithm to answer queries over a point set of size $\lceil w^{d\epsilon} \rceil$ and type 1 with $\sigma = \lceil w^\epsilon \rceil$ for any constant $0 < \epsilon < 1/d$ within $O(1)$ time and $O(\lg w)$ bits of space per point, which is presented in Lemma 4. This algorithm does not require a global lookup table, so it is able to handle larger word size $w = \omega(\lg n)$.

**Lemma 3.** *For any constant $0 < \epsilon < 1/d$, dominance counting queries over a point set $S$ of size $m \le n$ and type 1 with $\sigma = \lceil w^\epsilon \rceil$ can be supported using $O(m \lg w)$ bits of space and $O(1)$ query time.*

*Proof.* We sort all points of the point set in increasing order of the first coordinates, and divide the list into blocks of size $m_1 = w^2$. Then we divide each block into subblocks of size $m_2 = \lceil w^{d\epsilon} \rceil$. Each block/subblock is labeled with the largest first coordinate over the points inside the block/subblock. For each block $\beta$, we precompute a $d$-dimensional table $F_\beta$ in which, for $1 \le q_2, \ldots, q_{d+1} \le$

$\lceil w^\epsilon \rceil$, the entry $F_\beta[q_2, \ldots, q_{d+1}]$ stores the number of points in $S$ that are dominated by $(\texttt{label}(\beta), q_2, \ldots, q_{d+1})$, where $\texttt{label}(\beta)$ is the label of $\beta$. Similarly, for each subblock $\beta'$ of $\beta$, we maintain a $d$-dimensional table $g_{\beta'}$ in which, for $1 \leq q_2, \ldots, q_{d+1} \leq \lceil w^\epsilon \rceil$, the entry $G_{\beta'}[q_2, \ldots, q_{d+1}]$ stores the number of points inside $\beta$ that are dominated by $(\texttt{label}(\beta'), q_2, \ldots, q_{d+1})$.

Given a dominance counting query $q = (q_1, q_2, \ldots, q_{d+1})$, we find the rightmost block $\beta$ whose label is no greater than $q_1$. Then we find the rightmost subblock $\beta'$ to the right of $\beta$ whose label is no greater than $q_1$. Without loss of generality, we assume the existence of both $\beta$ and $\beta'$. The other cases can be handled similarly. Thus the answer to the given dominance counting query can be expressed as $F_\beta[q_2, \ldots, q_{d+1}] + G_{\beta'}[q_2, \ldots, q_{d+1}] + h$, where $h$ is the number of points in the subblock to the right of $\beta'$ that are dominated by the given query.

Later in Lemma 4, we will show the computation of $h$ requires $O(1)$ query time and $O(\lg w)$ bits of space per point. Thus, the overall query time for dominance counting queries over the point set of type 1 is $O(1)$. Finally we analyze the space cost. The tables for all blocks require $O((m/m_1) \times w^{d\epsilon} \times \lg m) = o(m)$ bits of space in total. The tables for all subblocks require $O((m/m_2) \times w^{d\epsilon} \times \lg m_1) = O(m \lg w)$ bits of space in total. Therefore the overall space cost is $O(m \lg w)$ bits. □

**Lemma 4.** *Dominance counting queries inside a subblock can be supported using $O(1)$ query time and $O(\lg w)$ bits of space per point.*

*Proof.* We divide a machine word into chunks of size $s_1 = d \cdot (\lceil \epsilon \lg w \rceil + 1)$ each. Each chuck is further divided into $d$ subchunks of size $s_2 = \lceil \epsilon \lg w \rceil + 1$ each. We sort all points in increasing order of the first coordinates, and, for each point in the point set, we store its second coordinate to its $(d+1)$-st coordinate in a chunk $\gamma$. For $1 \leq \ell \leq d$, the $(\ell+1)$-st coordinate will be stored in the $\ell$-th subchunk of $\gamma$. Note that these coordinates each fit in the lowest $\lceil \epsilon \lg w \rceil$ bits of a subchunk. The highest bit of the same subchunk, which is referred to as the flag bit, is set to be zero. Thus the space cost is $s_1 = O(\lg w)$ bits per point. Because each subblock consists of at most $m_2$ points and $m_2 \times s_1 = o(w)$, the chunks of all points in a subblock can fit in a single machine word.

Let $q = (q_1, q_2, \ldots, q_{d+1})$ be the query and $\beta'$ be the rightmost subblock that intersects with $q$. We find the rank $r$ of $q_1$ over the points of $\beta'$, and copy the chunks of the first $r$ points of $\beta'$ into the first $r$ chunks of a machine word $\mathcal{A}$. This requires only $O(1)$ time since these chunks are stored consecutively in memory. Then we store $q_2, q_3, \ldots, q_{d+1}$ duplicately in the first $r$ chunks of another word $\mathcal{B}$. For each of these $r$ chunks and each $1 \leq \ell \leq d$, $q_{\ell+1}$ is stored in the $\ell$-th subchunk as the lowest $\lceil \epsilon \lg w \rceil$ bits, and the flag bit of the subchunk is set to be 1. The construction of $\mathcal{B}$ also requires $O(1)$ time.

We then compute $\mathcal{C} = \mathcal{B} - \mathcal{A}$, mask all bits of $\mathcal{C}$ to 0 except the flag bits of the subchunks in each of the first $r$ chunks, and right-shift $\mathcal{C}$ by $s_2 - 1$ bits. It is not hard to see that a point is dominated by $q$ iff the value the corresponding chunk represents is equal to $(2^{ds_2} - 1)/(2^{s_2} - 1)$.

To count the occurrences of that value, we create another word $\mathcal{D}$ so that each of the first $r$ chunks represents $(2^{ds_2} - 1)/(2^{s_2} - 1) + 2^{ds_2 - 1}$. That is,

the lowest bits of all subchunks and the flag bit of the $d$-th subchunk are set to be 1 in each of the first $r$ chunks, and the other bit are set to be 0. We compute $\mathcal{E} = \mathcal{D} - \mathcal{C}$, and mask all bits of $\mathcal{E}$ to 0 except the highest bits of the first $r$ chunks, i.e., the flag bits of the $d$-th subchunks. The highest bit of a chunk is 1 iff the corresponding point is dominated by $q$.

Finally we sum up the highest bits of the first $r$ chunks. To achieve that, we right-shift $\mathcal{E}$ by $s_1 - 1$ bits, so that the highest bit of each chunk becomes the lowest one. Then we multiply the shifted word by $(2^{rs_1} - 1)/(2^{s_1} - 1)$ and the value stored in the $r$-th chunk will be the sum we need, which is also the answer to the query $q$. The whole algorithm requires $O(1)$ time and no table lookup. □

**Lemma 5.** *For any constant $0 < \epsilon < 1/d$, dominance counting queries over a point set $S$ of size $m \leq n$ and type $d'$ with $\sigma = \lceil w^\epsilon \rceil$ can be supported using $O(m \lg w \cdot (\lg n / \lg w + 1)^{d'-1})$ bits of space and $O((\lg n / \lg w + 1)^{d'-1})$ query time.*

*Proof.* The base case in which $d' = 1$ has been handled in Lemmas 3 and 4. We only show how to reduce the case of $d'$ to that of $d' - 1$. We build a range tree over the $d'$-th coordinates of points with branching factor $\lceil w^\epsilon \rceil$. The height of the range tree is bounded above by $O(\lg n / \lg w + 1)$. Each node $v$ in the range tree represents a range $[a_v..b_v]$ and the set $S(v)$ of points whose $d'$-th coordinates are in $[a_v..b_v]$. The leaf nodes in the range tree each represent a single point.

For each internal node $v$, we transform $S(v)$ into a point set $D(v)$ of type $d' - 1$. For any $\ell < d'$ and any point $p \in S(v)$, its $\ell$-th coordinate $p_\ell$ is replaced with the rank of $p_\ell$, i.e., the number of points in $S(v)$ whose $\ell$-th coordinates are no greater than $p_\ell$. In addition, the $d'$-th coordinate of $p$ is replaced with an integer in $[1..t]$, which is the index of $v$'s child that represents a set containing $p$. Queries over $D(v)$ can be supported recursively.

Inside each internal node $v$, for each dimension $1 \leq \ell \leq d'$ we write down a sequence $\mathcal{S}_{v,\ell}[1..|S(v)|]$. For each point $p \in S(v)$, $\mathcal{S}_{v,\ell}[p_\ell]$ is the integer that replaced the $d'$-th coordinate of $p$. We represent these sequences using the succinct data structures of Belazzougui and Navarro [1]. These data structures use $O(|\mathcal{S}_{v,\ell}| \lg w)$ bits of space, and support $\texttt{rank}_i(\mathcal{S}_{v,\ell}, p_\ell)$ operations in $O(1)$ time, which count the occurrences of $i$'s in $\mathcal{S}_{v,\ell}[1..p_\ell]$.

Let the given dominance counting query be $q = (q_1, q_2, \ldots, q_{d+1})$. Starting with the root node, we traverse the range tree from top to bottom. Let $v$ be the root node and let $v_1, \ldots, v_t$ be the children of $v$ from left to right. We find the largest $i$ so that $b_{v_i} \leq q_{d'}$. Querying $D(v)$ recursively with $(q_1, \ldots, q_{d'-1}, i, q_{d'+1}, q_{d+1})$, we can find the number of points in the first $i$ children of $v$ that are dominated by $q$. Then we recursively query $S(v_{i+1})$ with $q' = (q'_1, q'_2, \ldots, q'_{d'}, q_{d'+1}, \ldots, q_{d+1})$, where $q'_\ell = \texttt{rank}_{i+1}(\mathcal{S}_{v,\ell}, q_\ell)$ for $1 \leq \ell \leq d'$. We return the sum of the answers found.

This range tree is of height $O(\lg n / \lg w + 1)$, and a dominance counting query on a point set of type $d'$ is reduced to $O(\lg n / \lg w + 1)$ queries on points sets of type $d' - 1$. Thus we achieve the desired bounds for query time and space cost. □

**Remark.** Lemma 5 is a stronger version of Lemma 2(a). By Lemma 5, one can support $d$-dimensional range counting queries within $O((\lg n/\lg w+1)^{d-1})$ query time and $O(n\cdot(\lg n/\lg w+1)^{d-2})$ words of space. This improves the data structures of JaJa et al. [6] when $w \geq \lg^{\omega(1)} n$.

Next we consider how to prove Lemma 2(b). Unlike the structures for dominance counting queries with $\sigma = \lceil w^{\epsilon}\rceil$, for parallel counting queries we can only set $\sigma = \lceil \lg^{\epsilon} n\rceil$.

**Lemma 6.** *For any constant $0 < \epsilon < 1/d$, parallel counting queries over a point set $S$ of size $m \leq n$ and type 1 with $\sigma = \lceil \lg^{\epsilon} n\rceil$ can be supported using $O(m \lg\lg n)$ bits of space and $O(1)$ query time. In addition, we also need global lookup tables that occupy $o(n)$ bits of space in total.*

*Proof.* We sort all points of $S$ in increasing order of the first coordinates. We divide $S$ into blocks of size $n_1 = \lceil \lg^2 n\rceil$, and divide each block into subblocks of size $n_2 = \lceil \lg^{d\epsilon} n\rceil$. We still label each block/subblock with the largest first coordinate over the points inside the block/subblock.

For each block $\beta$ we maintain a table $D_\beta$ in which, for $j \in [0..\lceil(\lg n)/s\rceil]$ and $1 \leq q_2, q_3, \ldots, q_d \leq \lceil \lg^{\epsilon} n\rceil$, the entry $D_\beta[q_2, q_3, \ldots, q_d, j]$ stores the $j$-th sections of $f[1], f[2], \ldots, f[\lceil \lg^{\epsilon} n\rceil]$, where $f[i]$ is the number of points in $S$ that are dominated by $(\texttt{label}(\beta), q_2, \ldots, q_d, i)$. As described in Sect. 3, we store the $j$-th section of each of these values in a subword of $s+b+2$ bits, and pack them into a single word. These table $D_\beta$'s occupy $O(m/n_1)\times O(\lg^{(d-1)\epsilon} n)\times(\lceil \lg n/s\rceil+1)\times O(\lg n) = O(m/\lg^{1-d\epsilon} n) = o(m)$ bits in total.

For each subblock $\beta'$ of $\beta$ we maintain a table $E_{\beta'}$ in which, for $1 \leq q_2, q_3, \ldots, q_d \leq \lceil \lg^{\epsilon} n\rceil$, the entry $E_{\beta'}[q_2, q_3, \ldots, q_d]$ stores the concatenation of $g[1], g[2], \ldots, g[\lceil \lg^{\epsilon} n\rceil]$, where $g[i]$ is the number of points inside $\beta$ that are dominated by $(\texttt{label}(\beta'), q_2, \ldots, q_d, i)$. Each $g[i]$ can be represented in $\lceil \lg n_1\rceil = O(\lg\lg n)$ bits. The overall space cost for all the tables $E_{\beta'}$ is $O(m/n_2)\times\lceil \lg^{d\epsilon} n\rceil\times O(\lg\lg n) = O(m\lg\lg n)$ bits. We further precompute a global lookup table $X$ that, for each possible values of $g[1], g[2], \ldots, g[\lceil \lg^{\epsilon} n\rceil]$, stores a word in which the $i$-th subword is equal to $g[i]$. Clearly the lookup table $X$ requires $o(n)$ bits of space.

We can encode each subblock in $O(m_2 \lg\lg n) = O((\lg^{d\epsilon} n)\cdot\lg\lg n)$ bits. Then we precompute another global lookup table $Y$ that, for any possible encoding of a subblock $\beta'$ and any $1 \leq q_1, q_2, \ldots, q_d \leq \lceil \lg^{\epsilon} n\rceil$, stores the concatenation of $h[1], h[2], \ldots, h[\lceil \lg^{\epsilon} n\rceil]$, where $h[i]$ is the number of points inside $\beta'$ that are dominated by $(q_1, q_2, \ldots, q_d, i)$. The table $Y$ also requires $o(n)$ bits of space since there are only $O(n^{1-\delta})$ possible encodings of subblocks for some $\delta > 0$.

Let $(q_1, q_2, \ldots, q_d, j)$ be a parallel counting query. We find the rightmost block $\beta$ whose label is no greater than $q_1$, and the rightmost block $\beta'$ to the right of $\beta$ whose label is no greater than $q_1$. If $j > 0$, then we simply return $D_\beta[q_2, q_3, \ldots, q_d, j]$. If $j = 0$, then we further find the subblock $\beta''$ to the right of $\beta'$ and the rank $r$ of $q_1$ inside $\beta''$. The answer is the sum of $D_\beta[q_2, q_3, \ldots, q_d, 0]$, $E_{\beta'}[q_2, q_3, \ldots, q_d]$, and $Y[\texttt{enc}(\beta''), q_2, q_3, \ldots, q_d]$, where $\texttt{enc}(\beta'')$ is the encoding of $\beta''$. Note that we need $X$ to transform the entry of $E_{\beta'}$. The overall query time is $O(1)$. □

Finally, following the same approach of the proof for Lemma 5 but using branching factor $\lceil \lg^{\epsilon} n \rceil$, we can prove Lemma 2(b). Since the answer is expressed as a sum of the $j$-th sections of $K = O((\lg n / \lg\lg n)^{d'})$ numbers, we need to set $b$ larger than $\lg K = \Theta(\lg\lg n)$.

**Acknowledgements.** We thank the anonymous reviewers for their fruitful comments and suggestions.

## References

1. Belazzougui, D., Navarro, G.: Optimal lower and upper bounds for representing sequences. ACM Trans. Algorithms (TALG) **11**(4), 31:1–31:21 (2015). Article 31
2. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. Theor. Comput. Sci. **412**(24), 2588–2601 (2011)
3. Chan, T.M., Wilkinson, B.T.: Adaptive and approximate orthogonal range counting. In: SODA, pp. 241–251 (2013)
4. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. **47**(3), 424–436 (1993)
5. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: STOC, pp. 135–143 (1984)
6. JáJá, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 558–568. Springer, Heidelberg (2004)
7. Jørgensen, A.G., Larsen, K.G.: Range selection and median: tight cell probe lower bounds and adaptive data structures. In: SODA, pp. 805–813 (2011)
8. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. Nord. J. Comput. **12**(1), 1–17 (2005)
9. Matousek, J.: Reporting points in halfspaces. Comput. Geom. **2**, 169–186 (1992)