Fully Scalable MPC Algorithms for WSPD in Doubling and Euclidean Spaces

Eunjin Oh[1]
Department of Computer Science and Engineering, POSTECH, Pohang, South Korea

Hyeonjun Shin[1]
Department of Computer Science and Engineering, POSTECH, Pohang, South Korea

Abstract

In this paper, we study the problem of constructing a $(1/\varepsilon)$-*well-separated pair decomposition* (WSPD) for a point set of size n in the Massively Parallel Computation (MPC) model, where multiple machines work in parallel and communicate in synchronous rounds. We present an $O(1)$-round MPC algorithm that constructs a $O(1/\varepsilon)$-WSPD of size $(1/\varepsilon)^{O(\mathsf{ddim})} \cdot \tilde{O}(n)$ for point sets in a metric space of a constant doubling dimension ddim, with high probability, using $(1/\varepsilon)^{O(\mathsf{ddim})} \cdot \tilde{O}(n)$ total space and $O(n^{\delta})$ space per machine for a constant $\delta \in (0, 1)$. In the d-dimensional Euclidean space, we can improve the size of the WSPD and the total space to $(1/\varepsilon)^{O(d)} n$. This improves the best-known algorithm [FOCS'93] for computing a WSPD which requires $O(\log n)$ rounds and works only in Euclidean spaces. As a consequence, the following problems can be solved in $O(1)$ rounds in the MPC model: computing a $(1+\varepsilon)$-spanner, a $(1-\varepsilon)$-approximation of the diameter, the closest pair, and the k-nearest neighbors (k-NN). While our k-NN algorithm is specific to Euclidean space, the other three problems can be solved in both Euclidean and doubling metric spaces.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases MPC model, parallel algorithms, Computational Geometry, well-separated pair decomposition, doubling metric spaces

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.88

Related Version *Full Version*: `https://arxiv.org/abs/2607.03811`

Funding Supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00440239, Sublinear Scalable Algorithms for Large-Scale Data Analysis) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00358505).

1 Introduction

In this paper, we study geometric proximity problems in the *Massively Parallel Computation* (MPC) model, introduced by [37], now a standard framework for large-scale data processing in distributed systems such as MapReduce [22], Dryad [35], Hadoop [42], and Spark [43]. Here, the input data is distributed across multiple machines, and computation proceeds in synchronous rounds of local computation and communication. A central algorithmic goal is to minimize the number of communication rounds and to use small local memory while keeping the total work nearly linear. Over the past decade, significant progress has been made on fundamental MPC problems such as sorting, connectivity, and matching [7, 8, 10, 16, 20, 24, 27, 31, 38, 40]. While some geometric problems such as clustering, Euclidean MST, and spanners have been studied [4, 6, 9, 17, 21, 25, 36], they represent only a limited

[1] All authors contributed equally.

portion of the rich landscape of computational geometry. In particular, only a few prior works consider fundamental proximity structures such as well-separated pair decompositions, which are central to many classic geometric algorithms.

While only a few papers explicitly address geometric proximity problems in the MPC model, such problems have been extensively studied in earlier parallel computation models, particularly the PRAM model [3, 5, 13, 15, 19, 41]. For additional references, see the survey [30]. These algorithms typically assume global memory access and fine-grained synchronization, but these assumptions are less realistic in modern distributed systems. In contrast, the MPC model more accurately captures the constraints of today's large-scale computing environments by modeling limited local memory and emphasizing communication-efficient computation. Prior work [31, 37] has shown that PRAM algorithms can be simulated in the MPC model under reasonable assumptions. These simulation results can be viewed as implicit progress toward solving proximity problems in the MPC model. However, since a PRAM algorithm with depth t requires $O(t)$ rounds to simulate in the MPC model [31, 37], applying this approach to the aforementioned algorithms would still incur $\Omega(\log n)$ rounds. This highlights a fundamental challenge in designing native MPC algorithms that solve geometric proximity problems in sub-logarithmic rounds, fully leveraging the distributed nature of the model.

In this work, we take a systematic step in this direction by developing a fully scalable constant-round MPC algorithm for computing a well-separated pair decomposition in a metric space with a constant *doubling dimension* ddim, i.e., every ball of radius r can be covered by 2^{ddim} balls of radius $r/2$ [32]. We say that two sets A, B of points are $(1/\varepsilon)$-*separated* if $\max\{\mathsf{diam}(A), \mathsf{diam}(B)\} \leq \varepsilon \cdot \mathsf{dist}(A, B)$ for $\varepsilon > 0$. A collection $\mathcal{W} = \{\{A_1, B_1\}, \ldots, \{A_k, B_k\}\}$ is called a $(1/\varepsilon)$-*well-separated pair decomposition* ($(1/\varepsilon)$-WSPD) of a point set P if (i) $A_i, B_i \subset P$, and A_i and B_i are $(1/\varepsilon)$-separated for every i, (ii) $A_i \cap B_i = \emptyset$ for every i, and (iii) $\bigcup_{i=1}^{k} A_i \otimes B_i = P \otimes P$, where $A \otimes B = \{(x, y) \mid x \in A, y \in B, x \neq y\}$. We define the *size* of the WSPD as the number of pairs it contains. A WSPD compactly represents all pairs of point subsets that are sufficiently far apart, which has numerous applications. Due to its importance, computing a WSPD was among the earliest topics studied in parallel computational geometry under the PRAM model. For example, Callahan and Kosaraju [13] gave an optimal PRAM algorithm that computes an $O((1/\varepsilon)^d n)$-sized $(1/\varepsilon)$-WSPD of n points in d-dimensional Euclidean space in $O(\log n)$ depth using $O(n)$ processors. This yields an $O(\log n)$-round MPC algorithm via standard simulation [31, 37], but it remains open whether a $o(\log n)$-round MPC algorithm is possible. In contrast, in doubling metric spaces, a linear-sized WSPD is also known to exist [34], yet no efficient PRAM algorithm is known for constructing it. This makes it particularly challenging to design MPC algorithms in doubling metric spaces.

Our main results. In this paper, we present the first $O(1)$-round, fully scalable MPC algorithm for computing a $(1/\varepsilon)$-WSPD in a doubling metric space as stated below.

▶ **Theorem 1.** *Given a set P of n points in a metric space with a constant doubling dimension ddim for any $\varepsilon > 0$, we can compute a $(1/\varepsilon)$-WSPD of P of size $(1/\varepsilon)^{O(\mathsf{ddim})} n \log^2 n$ in $O(1)$ rounds, using $(1/\varepsilon)^{O(\mathsf{ddim})} n \log^3 n$ total space and $O(n^\delta)$ local memory per machine for any $\delta \in (0, 1)$. The algorithm succeeds with $1 - 1/n^{\Omega(1)}$ probability.*

Since the d-dimensional Euclidean space has doubling dimension $O(d)$, we can construct a WSPD by applying Theorem 1. However, by exploiting several classical geometric tools specific to Euclidean spaces, we can compute a linear-sized WSPD deterministically.

▶ **Theorem 2.** *Given a point set P of size n in $\mathbb{R}^d$ for any constant $d \geq 1$ and for any $\varepsilon > 0$, one can construct a $(1/\varepsilon)$-WSPD of P of size $(1/\varepsilon)^{O(d)}n$ in $O(1)$ rounds, using $(1/\varepsilon)^{O(d)}n$ total space and $O(n^\delta)$ local space[2] per machine for any $\delta \in (0,1)$.*

Our result improves upon the simulated PRAM algorithm of [13] in two key aspects: it reduces the round complexity from $O(\log n)$ to $O(1)$, and extends beyond constant-dimensional Euclidean spaces to general metric spaces with constant doubling dimension.

Applications. As applications, we present fully scalable MPC algorithms for constructing a spanner, computing the closest pair, approximating the diameter, and constructing all k-nearest neighbors as stated below. Let P be a set of n points in a metric space $\mathcal{X}$, and for any $\varepsilon > 0$, let $f_{\mathcal{X}}(n, \varepsilon)$ be the size of a $(1/\varepsilon)$-WSPD of P computed by our MPC algorithms.

- **Closest pair:** One can construct the *exact* closest pair of P in $O(1)$ rounds, using $f_{\mathcal{X}}(n, \varepsilon)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$.
- **Diameter:** One can construct a $(1-\varepsilon)$-approximate diameter of P in $O(1)$ rounds, using $f_{\mathcal{X}}(n, \varepsilon)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$.
- **Spanner:** One can construct a $(1+\varepsilon)$-spanner of P with $f_{\mathcal{X}}(n, \varepsilon)$ edges in $O(1)$ rounds, using $f_{\mathcal{X}}(n, \varepsilon)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$.
- **All k-nearest neighbors:** One can construct k-nearest neighbors of each point of P in $O(k)$ rounds in total, using $k \cdot f_{\mathcal{X}}(n, \varepsilon)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$, if $\mathcal{X}$ is a Euclidean space.

Comparison with prior work. As these problems are fundamental in computational geometry, several results are known. Although no prior work explicitly studies the closest-pair or diameter problem in the MPC model, they can be handled by classical geometric techniques. We also note that the closest-pair problem reduces to all-nearest neighbors. Overall, except for a few Euclidean-specific cases, our results match or outperform the best known MPC algorithms. Moreover, our approach is *unified*: the same WSPD-based framework simultaneously yields algorithms for all the problems discussed below.

- **Closest pair:** To the best of our knowledge, no prior MPC algorithm is known for the closest pair problem in doubling metric spaces. In contrast, it can be solved in $\mathbb{R}^d$ with $d \leq 2$ in $O(1)$ rounds by constructing the Delaunay triangulation [28, 39].
- **Diameter:** A 2-approximation of the diameter can be found in $O(1)$ rounds by selecting an arbitrary point and computing its farthest neighbor. In $\mathbb{R}^d$ for fixed d, a $(1-\varepsilon)$-approximation follows from evaluating the maximum directional width over a $(1/\varepsilon)^{O(d)}$ net on the unit sphere [2], which can be implemented in $O(1)$ rounds in the MPC model.
- **Spanner:** To the best of our knowledge, no prior MPC algorithm is known for constructing spanners in doubling metric spaces. In contrast, for Euclidean spaces, t-spanners with $O(n^{1+1/t^2})$ edges can be constructed in $O(1)$ rounds [18, 25], as explicitly stated in [36]. In high-dimensional Euclidean spaces, they outperform ours in terms of round complexity, whereas our spanner has fewer edges in low-dimensional Euclidean spaces.
- **All nearest neighbors:** To the best of our knowledge, no prior MPC algorithm is known for solving the all-nearest neighbors problem in doubling metric spaces. In contrast, for Euclidean spaces, all *c-approximate* nearest neighbors can be computed in $O(1)$ rounds

[2] More precisely, the total space and local space are $(1/\varepsilon)^{O(d)} \cdot (dn)$ and $O((dn)^\delta)$, respectively, which ensures full scalability. Since d is treated as a constant, we omit the dependence on d.

with total space complexity $n^{1/c} \cdot \tilde{O}(n)$ [21]. In high-dimensional settings, this result outperforms ours in terms of total space complexity, whereas our algorithm computes *exact* solutions with lower space usage in low-dimensional Euclidean spaces. In such low-dimensional settings, the previously best-known algorithm for approximate nearest neighbors achieves $O(1)$ rounds with $O(n)$ total space [1]. Our algorithm improves upon this by computing exact nearest neighbors within the same round and space complexity.

- **All k-nearest neighbors.** The all k-nearest neighbors problem can be solved in the binary-fork model [12], which can be simulated in the MPC model. Its performance depends on the spread and the expansion of P, whereas our algorithm works for arbitrary point sets. If we simulate their algorithm in the MPC model on a point set with bounded spread and bounded expansion, the number of rounds is $O(n^\gamma)$ for a constant $\gamma < 1$. In contrast, our algorithm uses $O(k)$ rounds for any point set.

Additional contribution: New hierarchical decomposition. To construct a WSPD efficiently, we build a hierarchical decomposition of P, which is also our contribution that may be of independent interest. In doubling metric spaces, we introduce a new hierarchical structure called the *partition cover tree*, which can be built in $O(1)$ rounds. A central tool for handling doubling metric spaces is a *net tree* [34]. While its uncompressed variant can be built in $O(1)$ rounds [6], it remains open whether the *compressed* net tree, which is essential for handling point sets with unbounded spread, can be built in $O(1)$ rounds. Here, the spread of a point set is the ratio between its largest and smallest pairwise distances. *Our work bypasses this bottleneck by introducing the partition cover tree.* Although it does not satisfy the packing property or the invariant that radii halve at each level unlike the net-tree, it retains a key geometric feature of the compressed net-tree: any ball can be covered by a small number of cells with comparable diameter, and the size is linear only in the input size. We believe this is of independent interest and may find broader applications in proximity problems over doubling metrics. For example, this may serve as a foundation for data structures that approximate local regions using a small number of representations such as consistent hashing and locality-sensitive hashing [11].

In Euclidean spaces, we present an MPC algorithm for constructing a *compressed quadtree* of linear size in $O(1)$ rounds. Tree-based hierarchical decompositions such as split trees, *kd*-trees, BBD-trees, randomly shifted quadtrees, partition tree for range searching have been designed in the MPC or PRAM model [1, 6, 13, 15], and have been used in a wide range of parallel geometric algorithms. In a similar sprit, our compressed quadtree might serve as a foundational structure for other MPC algorithms for proximity problems.

2 Preliminaries

Consider a metric space $\mathcal{X}$. For any two points p and q, we denote their distance in $\mathcal{X}$ by $\mathsf{dist}_{\mathcal{X}}(p, q)$. For convenience, we use $\mathsf{dist}(p, q)$ for $\mathsf{dist}_{\mathcal{X}}(p, q)$ if the underlying metric space is clear from the context. We assume that one can compute $\mathsf{dist}_{\mathcal{X}}(p, q)$ in $O(1)$ time for any metric space $\mathcal{X}$. Let $B_{\mathsf{dist}}(x, r) = \{y \in \mathcal{X} \mid \mathsf{dist}_{\mathcal{X}}(x, y) \leq r\}$. For convenience, we use $B(x, r)$ to denote $B_{\mathsf{dist}}(x, r)$ if the underlying distance function is clear from the context. And, for a ball B, we use $c(B)$ and $r(B)$ to denote the center and radius of B, respectively. Let P be a finite point set in the metric space $\mathcal{X}$. For any two subsets $X, Y \subseteq P$, we define their distance as the minimum distance between any pair of points $x \in X$ and $y \in Y$. That is, $\mathsf{dist}(X, Y) = \min_{x \in X, y \in Y} \mathsf{dist}(x, y)$. When $X = \{x\}$ is a singleton, we alternatively use $\mathsf{dist}(x, Y)$ to denote $\mathsf{dist}(X, Y)$. We denote the *diameter* of P, defined as the maximum pairwise distance between the points in P by $\mathsf{diam}(P) = \max_{p,q \in P} \mathsf{dist}(p, q)$.

Algorithm 1 genWSPD(u, v).

Assume $r_u \le r_v$. Otherwise, exchange the roles of u and v.
if $\max\{r_u, r_v\} \le (\varepsilon/8) \cdot \mathsf{dist}(p_u, p_v)$ **then**
    Return $\{u, v\}$
else
    Denote the children of v by $v_1, \dots v_i$.
    Return $\bigcup_{j=1}^{i} \mathsf{genWSPD}(u, v_j)$

In this paper, we consider two fundamental metric spaces: doubling metric spaces and Euclidean space. The doubling dimension $\mathsf{ddim}(\mathcal{X})$ of a metric space $\mathcal{X}$ is the smallest value d such that every ball B can be covered by 2^d balls of half the radius of B. A useful consequence of the doubling property is the packing property: any set of points in $B(x, r)$ that are pairwise at distance greater than r' has size at most $(r/r')^{O(\mathsf{ddim})}$.

Throughout this paper, we consider several tree structures. Let u be a node in a tree, and ℓ_u be the level of u, which is the distance from u to the root of the tree. We denote the parent of u by $\mathsf{par}(u)$. We use $[m]$ to denote $\{1, 2, \dots, m\}$ for an integer $m \ge 1$.

2.1 Well-Separated Pair Decomposition

In a metric space with doubling dimension ddim, any point set of size n has a $(1/\varepsilon)$-well-separated pair decomposition consisting of $(1/\varepsilon)^{O(\mathsf{ddim})}n$ pairs [15, 34]. While a doubling metric space has a WSPD of linear size, the sum of the sizes of all point pairs in the WSPD can be quadratic. To avoid explicitly listing all points in each pair, we use a compact representation: we represent each set as an arbitrary representative point in the set and the maximum distance from its representative point to any point in the set.[3]

Construction. We can compute a $(1/\varepsilon)$-WSPD efficiently once we have a *partition tree*. In the Euclidean case, one can use a quadtree as the partition tree [14, 15, 33], while in a doubling metric space, a net tree can serve as the partition tree [34]. We define a *partition tree* as a tree satisfying the following properties: (1) each node v corresponds to a subset P_v (called a *piece*) of P, (2) the pieces of the children of a node v form a partition of P_v, (3) the piece of the root is P, and (4) the piece of a leaf consists of a single point. Each vertex v is associated with a representative p_v and the maximum distance r_v from p_v to any point in P_v.

Algorithm 1 describes how to compute a WSPD using a partition tree T. In particular, we run Algorithm 1 with a pair consisting of two copies of the root of T. This algorithm always returns a $(1/\varepsilon)$-WSPD, but its size depends on the properties of the underlying partition tree. More specifically, the algorithm constructs a $(1/\varepsilon)$-WSPD by recursively checking whether two nodes satisfy the following, starting from the root node paired with itself.

$$\max\{r_u, r_v\} \le (\varepsilon/8) \cdot \mathsf{dist}(p_u, p_v). \tag{1}$$

If this holds, returns the pair $\{u, v\}$ as a pair of the WSPD; otherwise, assuming without loss of generality that $r_u \le r_v$, it recursively refines the pair by replacing v with its children: for each child v' of v, the algorithm recurses on the pair $\{u, v'\}$.

[3] In the Euclidean case, we construct a WSPD using a compressed quadtree. Since each set corresponds to a cell in the quadtree, we can simply use the center and diameter of that cell to represent this set.

2.2 Massively Parallel Computation (MPC) Model

In the MPC model, an input of size n is arbitrarily distributed over m machines, each with a local memory of $O(s)$ words. The number of machines is typically $m = O(n/s)$ or slightly larger. We say that an MPC algorithm is *fully scalable* if it works for $s = n^\delta$ with an arbitrary constant $\delta \in (0, 1)$. We assume that a word can store a real number with arbitrary precision as in [1], which is a standard assumption for geometric problems. We assume that each machine has a unique ID from $[m]$. The computation of an MPC algorithm proceeds in *synchronous rounds*, each consisting of a *local computation phase* followed by a *communication phase*. In the computation phase, each machine performs local computations on its local data. In the communication phase, each machine sends messages to other machines. Each machine is allowed to send and receive a total of $O(s)$ words per round.

Throughout this paper, we use the following subroutines that can be implemented in $O(1)$ MPC rounds: sorting, predecessor, indexing, minimum/maximum, broadcast. For an integer a, let M_a be the machine with ID a. The following subroutines can be implemented in $O(1)$ rounds in the MPC model.

1. **Sorting [29, 31]:** Given n comparable items distributed arbitrarily across the machines, one can sort and index all items in $O(1)$ rounds.
2. **Predecessor [7, 31]:** Given pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, distributed arbitrarily across the machines, where every item x_i has a total ordering and $y_i \in \{0, 1\}$, every item x_j can be stored together with the item x_{i_j} and its index i_j, where x_{i_j} is the largest item such that $x_{i_j} \leq x_j$ and $y_{i_j} = 1$.
3. **Indexing [7, 31]:** Suppose that the sets $S_1, S_2, \ldots, S_k$ of total n items are stored across the machines. In $O(1)$ rounds, every item x can be stored together with its rank within the set storing x in the machine that originally has x.
4. **Minimum/Maximum [23]:** Given two IDs a, b of the machines for $a \leq b$, the minimum or maximum of items stored in $M_a, M_{a+1}, \ldots M_b$ can be computed in $O(1)$ rounds.
5. **Broadcast [23]:** Given two IDs a, b of the machines for $a \leq b$, a machine can send a message of size $O(s^\alpha)$ to $M_a, M_{a+1}, \ldots M_b$ in $O(1)$ rounds, where $\alpha < 1$ is a constant.

2.3 Geometric Proximity Problems

We define the four fundamental problems that we address as applications of the WSPD. Let P be a point set in a metric space $\mathcal{X}$ with distance function dist.

- **Spanner:** A *t-spanner* of P is a compact representation of the distances between all the pairs of points of P. It is defined as a graph $G = (P, E)$ such that $\mathsf{dist}(q, s) \leq d_G(q, s) \leq t \cdot \mathsf{dist}(q, s)$, where $d_G(q, s)$ is the distance between q and s in G. Thus our goal in this problem is to compute a spanner with a small number of edges.
- **Diameter and closest pair:** The *diameter* and *closest pair* are fundamental geometric characteristics of a point set. Recall that the diameter of a point set P is defined as the maximum pairwise distance among the points, i.e., $\max_{p,q \in P} \mathsf{dist}(p, q)$. In this work, we aim to compute a $(1 - \varepsilon)$-approximate diameter, that is, the distance between some pair of points in P that is at least $(1 - \varepsilon) \cdot \mathsf{diam}(P)$. Conversely, the closest pair is the pair of distinct points in P with the minimum distance between them, i.e., a pair (x, y) such that $\mathsf{dist}(x, y) = \min_{p \neq q \in P} \mathsf{dist}(p, q)$.
- ***k*-nearest neighbors:** For a point $p \in P$, the *nearest neighbor* in P is the point in $P \setminus \{p\}$ that is closest to p. The *k-nearest neighbors* (k-NN) problem asks to compute, for each point $p \in P$, the k closest points in $P \setminus \{p\}$ to p. They are fundamental tasks in computational geometry and have numerous applications in clustering, classification, and data analysis.

Overview. In both doubling metric spaces and Euclidean spaces, our algorithm follows a unified two-step framework. First, we construct a partition tree T over the input point set P. Then, we generate pairs of nodes (u, v) in T such that the corresponding piece pair (P_u, P_v) form a WSPD. For doubling metric spaces, we introduce a new structure called the *partition cover tree*, specifically designed for our setting. In contrast, for Euclidean spaces, we use a compressed quadtree as the partition tree.

Due to space limitations, all omitted proofs and technical details are deferred to the full version.

3 WSPD Construction in Doubling Metric Spaces

Let P be a set of n points in a metric space with a constant doubling dimension ddim. We introduce our approach with three ingredients: *partition cover trees*, *radius perturbation*, and *α-approximation*. We compute a WSPD in two steps: first, by constructing a *partition cover tree* and then by constructing a WSPD using this tree. Given the partition cover tree, we can construct a WSPD by parallelizing Algorithm 1. Details can be found in the full version.

First ingredient: Partition cover tree. Given a partition tree T of P, we can compute a WSPD by traversing the tree as in sequential algorithm by Har-Peled and Mendel [34]. However, the size of the constructed WSPD depends on the properties of the underlying partition tree. For instance, Har-Peled and Mendel [34] used a net tree to ensure that the number of pairs in the WSPD is $O(n)$. However, it is unclear whether their algorithm can be adapted to the MPC model, as it relies on constructing a *greedy permutation* of P.[4]

Instead, we present a new type of partition tree that enables the construction of a WSPD of size $(1/\varepsilon)^{O(\mathsf{ddim})} n \log^2 n$. We show that any randomized partition tree satisfying the following property yields a WSPD of size $(1/\varepsilon)^{O(\mathsf{ddim})} n \log^2 n$. We call a partition tree a *partition cover tree* if it satisfies the following small cover property. Its proof can be found in the full version.

▶ **Property 3** (Small Cover Property). *For a ball B, a set $\{v_1, v_2, \ldots, v_k\}$ of nodes of a randomized partition tree T is called a* cover *of B induced by T if the union of $P_{v_1}, \ldots, P_{v_k}$ contains B, and $\mathsf{diam}(P_{v_i}) \le \mathsf{diam}(B)/10$ for all nodes v_i. We say T has the* small cover property *if for every ball B in the metric space $\mathcal{X}$, the expected size of the minimum-cardinality cover of B is $O(\log n)$.*

As mentioned earlier, a major obstacle in designing MPC algorithms for doubling metric spaces with unbounded spread has been the absence of an efficient MPC construction of the *compressed net tree*. Although the partition cover tree does not satisfy the packing property or the invariant that radii halve at each level, it shares the small cover property.

Second ingredient: Radius perturbation. We can construct a partition tree by partitioning P into subsets in a top-down manner. Here, we describe the construction in the sequential model, and later we briefly show how to implement it in the MPC model using α-approximation. Imagine that we compute a ball B containing $|P|/2$ points of P. Then we recursively construct the tree of $P \cap B$ and the tree of $P \setminus B$, and merge them into a single

[4] A greedy permutation is a sequence of points $\langle p_1, \ldots, p_n \rangle$ from P, associated with a sequence $\langle r_1, \ldots, r_n \rangle$ of radii, such that $P \subseteq \cup_{\ell=1}^{k} B(p_\ell, r_k)$, and $\min_{1 \le i \le j \le k} \mathsf{dist}(p_i, p_j) = r_{k-1}$ for any integer k. Here, the choice of each point p_i depends on the prefix $\langle p_1, \ldots, p_{i-1} \rangle$ of the sequence.

tree by creating the root corresponding to P. Clearly, the constructed tree is a partition tree of height $O(\log n)$, but it does not necessarily satisfy the small cover property. To see this, observe that a small ball can be split into two subsets by B. Moreover, these subsets can be further subdivided at higher levels. As a result, a small ball may be split into $\Theta(n)$ pieces before reaching levels where the diameters of the pieces become smaller than the diameter of the ball itself.

To avoid the case that a small ball is separated into many pieces, we choose the radius of a ball in the construction of the partition tree randomly. Instead of choosing a ball containing $|P|/2$ points of P, we first compute a smallest-radius ball B containing at least $|P|/4^{O(\mathsf{ddim})}$ points from P. Then $|P|/4^{O(\mathsf{ddim})} \leq |B(c, 2r) \cap P| \leq |P|/2^{O(\mathsf{ddim})}$ by the choice of B, where c and r denote the center and radius of B, respectively. The second inequality holds, since $B(c, 2r)$ can covered by at most $2^{O(\mathsf{ddim})}$ balls of radius slightly smaller than r. Thus, to ensure that a ball separates P in a balanced way, we may choose any radius from $[r, 2r]$. We choose a value θ from $[0, 1]$ uniformly at random, and partition P into two subsets using $B^* = B(c, (1 + \theta) \cdot r)$. Then the probability that a ball of radius r' is separated by B^* is at most r'/r, which is very small if $r' \ll r$. Then we recurse on $P \cap B^*$ and $P \setminus B^*$. For a technical reason, if $r \geq \bar{r}/10$, where $\bar{r}$ is the radius of the separating ball of its parent, we further partition P using an arbitrary ball of radius $\bar{r}$ with radius perturbation, and then construct a binary partition tree of this partition first, and then recurse on each piece of the partition. Then we can show that the resulting tree T satisfies the small cover property. Its proof can be found in the full version.

In the MPC framework based on α-approximation, which will be described later, it is crucial not only to obtain a partition satisfying the small cover property, but to obtain it as a *hierarchical* tree of *height* $O(\log n)$. This $O(\log n)$-height hierarchical structure is what enables us to construct the tree in $O(1)$ MPC rounds in a top-down manner. Without such constraints, one may satisfy the covering property in other ways like [26], but these constructions do not appear to admit $O(1)$-round parallelization in the MPC model.

Ensuring both hierarchy and bounded height, however, makes the analysis considerably more delicate. Although radius perturbation has appeared in prior geometric decompositions, those arguments rely on the property that the diameters of the pieces decrease by a constant factor along a root-to-leaf path. However, in our construction, the radii of the separating balls do not necessarily decrease by a constant factor along root-to-leaf paths, and therefore the standard arguments used in classical radius-perturbation schemes do not carry over. This is a main technical challenge we overcame in this paper. Due to space limitations, the full analysis and full MPC construction are deferred to the full version.

Third ingredient: α-approximation. To construct the partition cover tree T in $O(1)$ rounds, we compute the first $(\log s)/2$ levels of T at once using an $(1/s^{1/4})$-approximation of a proper range space, which is our third ingredient. A similar approach is also used in [1] to construct a range tree in the Euclidean space in the MPC model. An α-approximation is originally introduced for approximate counting and geometric estimation. Here, we review several notions for defining the α-approximation. See also [33] for further details. A *range space* S is a pair $(X, \mathcal{R})$, where X is a ground set, and $\mathcal{R}$ is a family of subsets of X, called *ranges*.

▶ **Definition 4** (α-approximation)**.** *Given a range space $(X, \mathcal{R})$ and a parameter $0 \leq \alpha \leq 1$, a subset $Y \subseteq X$ is an* α-approximation *of $(X, \mathcal{R})$ if, for every $R \in \mathcal{R}$,*

$$\left| \frac{|Y \cap R|}{|Y|} - \frac{|R|}{|X|} \right| \leq \alpha.$$

An α-approximation of a range space S can be obtained via random sampling, where the required sample size depends on the VC dimension of S. Or, alternatively, if the number of ranges is small, we can use the following lemma. Its proof can be found in the appendices.

▶ **Lemma 5** (Folklore). *There exists a constant $c > 0$ such that a set obtained by m random independent draws from X forms an α-approximation of $(X, \mathcal{R})$ with probability at least $1-\psi$, regardless of VC dimension, where*

$$m = \frac{c}{\alpha^2}\left(\log|\mathcal{R}| + \log\frac{1}{\psi}\right).$$

Proof. Let R be a fixed range of $\mathcal{R}$. Let $p_R = |R|/|X|$ be the true fraction of X in R. Let $Z_1, Z_2, \ldots, Z_m$ be the indicator random variables where $Z_i = 1$ if the ith sampled point is contained in R, and $Z_i = 0$, otherwise. Then each Z_i is a Bernoulli random variable with mean $\mathbb{E}[Z_i] = p_R$. Let $\hat{p}_R = (1/m)\sum_1^m Z_i$. By the additive Chernoff bound (or the Hoeffding's inequality), for any $\alpha > 0$, we have

$$\Pr[|\hat{p}_R - p_R| > \alpha] \leq 2e^{-2\alpha^2 m}.$$

That is, the success probability for R is high. Now we take a union bound over all ranges R on $\mathcal{R}$.

$$\Pr[\exists R \in \mathcal{R} \text{ s.t } |\hat{p}_R - p_R| > \alpha] \leq |\mathcal{R}| \cdot 2e^{-2\alpha^2 m}.$$

By the choice of m, we can show that the success probability is at least $1-\psi$. ◀

We use the α-approximation to parallelize the construction of a partition cover tree. Consider the range space $S = (P, \mathcal{R})$ where $\mathcal{R} = \{\cap_{B\in\mathcal{B}} B \setminus \cup_{B'\in\mathcal{B}'} B'\}$, and $\mathcal{B}$ and $\mathcal{B}'$ are sets of $O(\log n)$ balls. Since the size of $\mathcal{R}$ is $n^{O(\log n)}$, a sample Y of size $O(\sqrt{s}\log^2 n) \leq O(s)$ is an $(s^{1/4})$-approximation of S. Then we construct a partition tree of Y until every leaf corresponds to a set of $O(1)$ points of Y. This also partitions P into $O(s)$ subsets, each is contained in $\mathcal{R}$, accordingly. The fact that Y is an $(s^{1/4})$-approximation of S implies that each subset in the partition contains $O(n/s^{1/4})$ points of P. Therefore, by repeating this $O(1)$ times, we can obtain a partition cover tree in $O(1)$ rounds. See the full version for details.

▶ **Theorem 6.** *Given a set P of n points in a doubling metric space with bounded doubling dimension* ddim*, one can construct the partition cover tree of P of $O(n)$ size and complexity. It takes $O(1)$ rounds with high probability, and uses $O(n)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0, 1)$.*

Both the partition cover tree and the tree of Andoni et al. [6] can be viewed as variants of a net tree in the sense that they preserve several key structural properties of net trees. However, neither construction is a full-fledged net tree in the classical sense. A main advantage of our structure is that we can construct it in $O(1)$ MPC rounds even when *the spread is not bounded.* In the following, we discuss the precise relationship between the two structures, and which properties they share or differ in. Andoni et al. [6] also used the radius perturbation scheme to construct a tree with key structural properties of an *uncompressed* net-tree in the MPC model. Their construction assumes that the input point set has bounded spread. We highlight two main differences between their tree and our partition cover tree below.

- First, the desired properties of the resulting tree differ. The authors in [6] designed their tree as part of the MST construction. However, their analysis does not immediately guarantee that the tree satisfies the small cover property. In contrast, our construction

explicitly aims to achieve the small cover property, which requires new technical insights. This is particularly challenging in our setting because the radii of separating balls do not necessarily decrease by half along root-to-leaf paths. Such non-monotonicity arises when a node lies outside the separating ball of its ancestor. Establishing that our tree satisfies the small cover property is therefore a key technical contribution of this work.

- Second, the construction itself differs from theirs. The authors in [6] recursively subdivide each piece into $2^{O(\mathsf{ddim})}$ smaller pieces of approximately half the diameter. Given the assumption that the spread is polynomially bounded in n, the resulting tree has height $O(\log n)$, allowing its construction in $O(1)$ MPC rounds. However, in the general case without the bounded spread assumption, this method yields a tree of height $O(n)$, making the same constant-round parallelization infeasible. In contrast, our approach selects a separating ball B that balances the size of each piece. This strategy ensures that the resulting tree has height $O(\log n)$ regardless of the spread, enabling its construction in $O(1)$ MPC rounds via an α-approximation, as described later.

Constructing a WSPD. Given a partition cover tree T, we construct a WSPD of size $(1/\varepsilon)^{O(\mathsf{ddim})} n \log^2 n$ with at least constant probability. We use the observation from [33] for Euclidean spaces with a compressed quadtree as the partition tree: for any pair (u, v) in the WSPD constructed by Algorithm 1, such that the recursion on $\{u, v\}$ is invoked from the recursion on $\{u, \mathsf{par}(v)\}$, the following conditions hold. $\mathsf{dist}(A_{\mathsf{par}(u)}, A_{\mathsf{par}(v)}) \leq (1/\varepsilon) \cdot \mathsf{diam}(A_{\mathsf{par}(v)})$, and $\mathsf{diam}(A_{\mathsf{par}(v)}) \leq \mathsf{diam}(A_{\mathsf{par}(u)})$, where A_u is the piece corresponding to u. We show that analogous conditions hold for doubling metric spaces when the partition tree is the partition cover tree: for any pair $\{u, v\}$ in the WSPD constructed by Algorithm 1, $\mathsf{dist}(p_v, p_{\mathsf{par}(u)}) \leq (8/\varepsilon) r_{\mathsf{par}(u)}$, and $r_{\mathsf{par}(u)} \leq r_{\mathsf{par}(v)}$. Its proof can be found in the full version. We refer to a node $u \in T$ as a *pairing candidate* of a node $v \in T$ if v satisfies these conditions.

To construct a WSPD, we identify the *candidate pairs* consisting of a node in T and one of its pairing candidates. We perform this efficiently in parallel by grouping $O(\log s^{1/4})$ consecutive levels of T. Within this range of levels, the nodes induce several disjoint subtrees of T. For each pair of subtrees, we identify all candidate pairs formed by one node from each subtree. This computation fits in a single machine, since every subtree in the range has size at most $O(s^{1/4})$. However, the total number of candidate pairs from a pair of subtrees may exceed local memory. To avoid this, we utilize the pre-assignment strategy: we first count the number of such pairs and allocate a sufficient number of machines to construct them. Furthermore, we can amplify the success probability to $1 - 1/n^{\Omega(1)}$ by increasing the total space by a factor of $O(\log n)$, without increasing the local memory. It concludes Theorem 1.

4 WSPD Construction in Euclidean Space

Now we consider a set P of n points in $\mathbb{R}^d$. As in the doubling metric case, we compute a WSPD in two steps: first, by constructing a partition tree, and then by constructing a WSPD using this tree. However, instead of the partition cover tree, we construct a compressed quadtree, which leads to an improvement in the size of the WSPD. Given the compressed quadtree, we can construct a WSPD similarly to the doubling metric case, while the process is considerably simpler in the Euclidean case. We can assume that $P \subseteq [0, 1)^d$, since all coordinates can be uniformly shifted and scaled to fit within $[0, 1)$, which can be done in $O(d) = O(1)$ rounds by computing and broadcasting the maximum value of each coordinate.

As in [33], we use a compressed quadtree to construct a WSPD. Consider the axis-aligned unit hypercube $\square$ containing P. The unit hypercube can be hierarchically partitioned into smaller axis-aligned hypercubes, where each cell is recursively subdivided into 2^d cells of

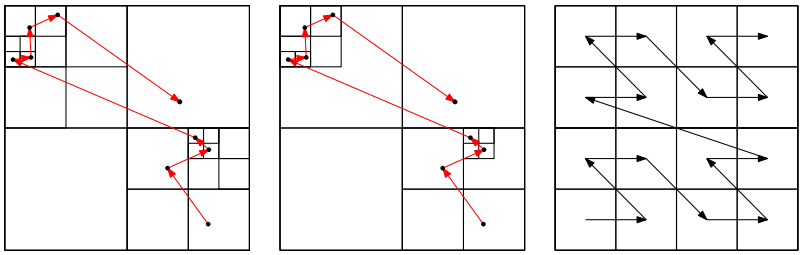

Figure 1 (a) Illustration of the quadtree. (b) Illustration of the compressed quadtree. In (a) and (b), the ordering of the points colored red is the $\mathcal{Z}$-order. (c) Illustration of the $\mathcal{Q}$-order among the cells of the same side length.

half the side length until every cell contains at most one point in P. The *(regular) quadtree* of P is a tree T whose nodes correspond to a cell of the hierarchical partition of $\square$, and whose non-leaves have 2^d children, satisfying the following properties. Let $\square_v$ denote the corresponding cell of a node v of T. The cells corresponding to the children of a non-leaf u form a subdivision of $\square_u$ into cells of half the side length. Each leaf of T corresponds to a cell containing at most one point in P. Throughout this paper, if $\square_v$ of a leaf v contains a point $p \in P$, we consider $\square_v$ as p itself.

Note that the height of the quadtree can be unbounded in the worst case if the spread of P is not bounded. Thus, instead of a quadtree, we need to use a *compressed quadtree*, which is a compact representation of the quadtree. Intuitively, the compressed quadtree of P is the tree whose nodes are the nodes in a quadtree of P whose corresponding cells contain at least one point in P, and no child is corresponding to a cell containing the same subset of P. More specifically, the compressed quadtree of P is obtained from the (regular) quadtree by removing all nodes whose cells are empty (i.e., contain no point of P) and contracting any node whose cell contains the same point set as its only child. Observe that each leaf of the compressed quadtree corresponds to a unique point in P.

Key idea: Using $\mathcal{Q}$-order and $\mathcal{Z}$-order. To construct the compressed quadtree, we make use of a total ordering of cells of T and points of P; see [33]. Consider a quadtree T of P, and its depth-first search (DFS) traversal, where the children of each node are always visited in the same pre-defined order. The *$\mathcal{Q}$-order* of T is a total ordering of the cells corresponding to nodes in T according to the DFS order of T. See Figure 1(c). Notice that the $\mathcal{Q}$-order on the compressed quadtree is inherited directly from its underlying (regular) quadtree. Moreover, the total ordering of P obtained by restricting the $\mathcal{Q}$-order to the points of P is known as the *$\mathcal{Z}$-order* of P. See Figure 1(a-b).

As in [33], we assume that both the floor operation $\lfloor \cdot \rfloor$ and the bit-index operation $\mathrm{bit}_\Delta(\alpha, \beta)$ can be computed in constant time, where $\mathrm{bit}_\Delta(\alpha, \beta)$ denotes the index of the first bit (after the radix point) at which α and β differ when represented in binary. Under this assumption, both $\mathcal{Q}$- and $\mathcal{Z}$-order comparisons can be performed in $O(1)$ time, using only the coordinates of the centers of cells for the $\mathcal{Q}$-order and point coordinates for the $\mathcal{Z}$-order. Similarly, for any two points in P, the smallest cell in the compressed quadtree that contains both can be computed in constant time using only their coordinates. Furthermore, given a point and a side length, we can compute in constant time the cell in the quadtree of that side length containing the point.

MPC construction the compressed quadtree. We first generate all nodes of T, and then identify their parents and children. We begin by sorting all points in P according to the $\mathcal{Z}$-order. This can be done efficiently in parallel, as $\mathcal{Z}$-order comparisons rely only on the coordinates of the points. Next, for each pair of consecutive points under the $\mathcal{Z}$-order, we construct the smallest cell containing both points. Each such cell corresponds to a non-leaf of the compressed quadtree. Then, we have all nodes in the compressed quadtree since each point corresponds to a unique leaf. Subsequently, we compute the parent and children for each node in the compressed quadtree. In particular, for each $1 \leq i \leq 2^d$, we sort the cells, so that the cell corresponding to a node and that of its i-th child under the DFS order appear consecutively. This can be done by modifying the pre-defined order among siblings, which defines the $\mathcal{Q}$-order. Consequently, this allows us to associate each node with its i-th child in parallel. We repeat this for all $1 \leq i \leq 2^d$ to associate every node with its children.

Constructing a WSPD. As for doubling metric spaces, we have the following observation from [33]: For a pair $(u, v) \in \mathcal{W}$ constructed by Algorithm 1 such that the recursion on $\{u, v\}$ is invoked from the recursion on $\{u, \mathsf{par}(v)\}$, $\mathsf{dist}(\square_{\mathsf{par}(u)}, \square_{\mathsf{par}(v)}) \leq (1/\varepsilon) \cdot \mathsf{diam}(\square_{\mathsf{par}(v)})$, and $\mathsf{diam}(\square_{\mathsf{par}(v)}) \leq \mathsf{diam}(\square_{\mathsf{par}(u)})$.[5] Here, we refer to a node $u \in T$ as a pairing candidate of a node $v \in T$ if u satisfies the above bounds. Then, we identify all pairing candidates for each node in T. To do this, we construct auxiliary cells for each node v whose side length equals that of $\square_{\mathsf{par}(v)}$, which align with the hierarchical partition defining T and lie within distance $(1/\varepsilon) \cdot \mathsf{diam}(\square_{\mathsf{par}(v)})$ from $\square_{\mathsf{par}(v)}$. We then sort all auxiliary cells and all nodes in the $\mathcal{Q}$-order of their corresponding cells, breaking ties by letting the auxiliary cell precede the node corresponding to the identical cell. Observe that the successor node of an auxiliary cell of v or its children may be a pairing candidate for the children of v. Thus, we can identify all pairing candidates for each node in parallel. It concludes Theorem 2.

▶ **Theorem 7.** *Given a set P of n points in $\mathbb{R}^d$ for a constant d, one can construct the compressed quadtree of P of $O(n)$ size and complexity in $O(1)$ rounds, using $O(n)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0, 1)$.*

5 Applications of WSPD

In this section, we present four applications: fully scalable MPC algorithms for constructing a spanner, computing the closest pair, approximating the diameter, and computing all k-nearest neighbors. The first three follow the framework of [33], whereas the k-nearest neighbors algorithm adopts the cone-based decomposition which is specific to the Euclidean spaces.

Spanner, Diameter Approximation, and Closest pair. The following results are immediate consequences of our MPC construction of WSPD [33]. To construct a $(1+\varepsilon)$-spanner, we first construct a (c/ε)-WSPD $\mathcal{W}$ with $c \geq 16$. Then, a $(1+\varepsilon)$-spanner can be obtained by forming a graph by constructing edges between the representative points of each pair of $\mathcal{W}$. Therefore, we only need to compute the distance between the representative points of each pair of $\mathcal{W}$. This step requires only local computation, as all pairs in the $\mathcal{W}$ are explicitly stored together with their representative points.

[5] To make the description consistent with [33], instead of using r_v in Algorithm 1, we use $\mathsf{diam}(\square_v)$ to measure the diameter of the point set corresponding to v, and we use $\mathsf{dist}(\square_u, \square_v)$ to measure the distance between two points from $\square_u$ and $\square_v$.

▶ **Theorem 8.** *Let P be a set of n points, and let $\varepsilon > 0$. If P is in $\mathbb{R}^d$ for constant d, one can construct a $(1+\varepsilon)$-spanner of P with $(1/\varepsilon)^{O(d)}n$ edges in $O(1)$ rounds, using $(1/\varepsilon)^{O(d)}n$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$. If P is in a metric space of doubling dimension ddim for constant ddim, one can construct a $(1+\varepsilon)$-spanner of P with $(1/\varepsilon)^{O(\mathsf{ddim})}n\log^2 n$ edges in $O(1)$ rounds with probability $1-1/n^{\Omega(1)}$, using $(1/\varepsilon)^{O(\mathsf{ddim})}n\log^3 n$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$.*

For $(1-\varepsilon)$-approximate diameter, we only need to compute the maximum distance between the representative points in each pair of a $(4/\varepsilon)$-WSPD. More specifically, we first construct a $(4/\varepsilon)$-WSPD $\mathcal{W}$, and compute the distance between the representative points of each pair of $\mathcal{W}$ as for the previous problem. After that, we compute the maximum of them.

▶ **Theorem 9.** *Let P be a set of n points, and let $\varepsilon > 0$. If P is in $\mathbb{R}^d$ for constant d, one can compute $(1-\varepsilon)$-approximate diameter of P in $O(1)$ rounds, using $(1/\varepsilon)^{O(d)}n$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$. If P is in a metric space of doubling dimension ddim for constant ddim, one can compute $(1-\varepsilon)$-approximate diameter of P in $O(1)$ rounds with probability $1-1/n^{\Omega(1)}$, using $(1/\varepsilon)^{O(\mathsf{ddim})}n\log^3 n$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$.*

Similarly, we can find the exact closest pair. We first construct a 2-WSPD $\mathcal{W}$, and compute the distance between the representative points of each pair of $\mathcal{W}$ as for the previous problem. After that, we compute the minimum of them.

▶ **Theorem 10.** *Let P be a set of n points, and let $\varepsilon > 0$. If P is in $\mathbb{R}^d$ for constant d, one can compute the closest pair in $O(1)$ rounds, using $O(n)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$. If P is in a metric space of doubling dimension ddim for constant ddim, one can compute the closest pair in $O(1)$ rounds with probability $1-1/n^{\Omega(1)}$, using $O(n\log^3 n)$ total space and $O(n^\delta)$ local space per machine for any $\delta \in (0,1)$.*

k-Nearest Neighbors in $\mathbb{R}^d$. Now, we present our $O(k)$-round MPC algorithm for constructing all k-nearest neighbors for a set P of n points in $\mathbb{R}^d$ for constant d using $O(kn)$ total space and $O(kn^\delta)$ local space per machine for any $\delta \in (0,1)$. Our algorithm is based on the approach of Callahan and Kosaraju [15] introduced for a $O(\log n)$-depth PRAM algorithm. Now let T be a compressed quadtree, and $\mathcal{W}$ be the 3-WSPD constructed from T.

The key idea of [15] is to prune the search space, so that the problem is reduced to computing the *search space* $N(p,\mathbf{u})$ for all points p in P and the constant number of directions $\mathbf{u}$. Here, $N(p,\mathbf{u})$ is defined as follows. Let U be a finite set of direction vectors in $\mathbb{R}^d$ such that every direction lies within an angle of at most $1/2$ radian from some vector in U of size $O(2^d) = O(1)$. For a direction $\mathbf{u} \in U$, the *search set* of a node v is defined as the set of k nearest points to o_v among all points that lie in some cell $\square \cap \mathcal{C}(o_v,\mathbf{u})$ paired with $\square_v$ in $\mathcal{W}$, such that $|P \cap \square| \le k$, where $\mathcal{C}(p,\mathbf{u})$ is the cone with apex $p \in P$, axis in direction $\mathbf{u} \in U$, and opening angle at most $1/2$ radian. Let v_p be the leaf of T containing $p \in P$, and let $v_{p,\mathbf{u}}$ be the lowest ancestor of v_p such that the union of the search sets for a direction $\mathbf{u} \in U$ of all nodes on the path from v_p to $v_{p,\mathbf{u}}$ in T contains at least k points. Then, $N(p,\mathbf{u})$ is defined as the set of the k nearest points of p in the union of the search sets of all nodes on the path from v_p to the $(\ell_{v_p} - \lambda)$-level ancestor of $v_{p,\mathbf{u}}$, where λ is a constant specified in [15], and ℓ_{v_p} denotes the *level* of v_p, i.e., the distance from v_p to the root of T.

We now describe an algorithm for computing all $N(p,\mathbf{u})$ in the MPC model, which is our own contribution. We first construct the search set for every node v in T and every direction $\mathbf{u} \in U$. To do this, for each node v of T such that $P \cap \square_v \le k$, we maintain all points in

$\square_v$, before constructing $\mathcal{W}$. After that, we find $v_{p,\mathbf{u}}$ for every point p. Observe that there might be many nodes in the path in T between v_p and $v_{p,\mathbf{u}}$ for a point p and a direction $\mathbf{u}$, since some nodes have empty search sets. Thus, it is unclear if one can compute $v_{p,\mathbf{u}}$ in $O(1)$ rounds. To avoid this case, we construct a new tree $T_{\mathbf{u}}$ consisting of the nodes of T whose search sets are non-empty. We then find $v_{p,\mathbf{u}}$ for all points p in P in parallel by propagating the search set of each node through its descendants in $T_{\mathbf{u}}$. After that, we construct $N(p,\mathbf{u})$ for every point $p \in P$ in a similar manner.

Note that if p is one of the k-nearest neighbors of a point q, then q is contained in the union of $N(p,\mathbf{u})$ over all $\mathbf{u} \in U$ [15]. Therefore, if we have the union of $N(p,\mathbf{u})$ for every point p, then we can find all k-nearest neighbors in $O(k)$ rounds using Sorting and Minimum operations.

▶ **Theorem 11.** *Given a set P of n points in $\mathbb{R}^d$ for constant d, one can compute k nearest neighbors of all points of P in $O(k)$ rounds, using $O(kn)$ total space and $O(kn^\delta)$ local space per machine for any $\delta \in (0,1)$. For a given point $p \in P$, finding k-nearest neighbors of p takes $O(1)$ rounds.*

References

1. Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, and Abhinandan Nath. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2016)*, pages 429–440, 2016. doi:10.1145/2902251.2902303.
2. Pankaj K Agarwal, Jiří Matoušek, and Subhash Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry*, 1(4):189–201, 1992.
3. Alok Aggarwal, Bernard Chazelle, Leo Guibas, Colm Ó'Dúnlaing, and Chee Yap. Parallel computational geometry. *Algorithmica*, 3(1):293–327, 1988. doi:10.1007/BF01762120.
4. AmirMohsen Ahanchi, Alexandr Andoni, MohammadTaghi Hajiaghayi, Marina Knittel, and Peilin Zhong. Massively parallel tree embeddings for high dimensional spaces. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2023)*, pages 77–88, 2023. doi:10.1145/3558481.3591096.
5. Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pages 683–694, 1994. doi:10.1109/SFCS.1994.365724.
6. Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 574–583, 2014. doi:10.1145/2591796.2591805.
7. Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 674–685, 2018. doi:10.1109/FOCS.2018.00070.
8. Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1616–1635, 2019. doi:10.1137/1.9781611975482.98.
9. Amir Azarmehr, Soheil Behnezhad, Rajesh Jayaram, Jakub Łącki, Vahab Mirrokni, and Peilin Zhong. Massively parallel minimum spanning tree in general metric spaces. In *Proceedings of the 36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*, pages 143–174, 2025.
10. Alkida Balliu, Rustam Latypov, Yannic Maus, Dennis Olivetti, and Jara Uitto. Optimal deterministic massively parallel connectivity on forests. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2589–2631, 2023. doi:10.1137/1.9781611977554.CH99.

11 Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via lsh based data partitioning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 570–579, 2018.

12 Guy E. Blelloch and Magdalen Dobson. Parallel nearest neighbors in low dimensions with batch updates*. In *Proceedings of the 24th Symposium on Algorithm Engineering and Experiments (ALENEX 2022)*, pages 195–208, 2022. doi:10.1137/1.9781611977042.16.

13 Paul B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS 1993)*, pages 332–340, 1993.

14 Paul B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications*. The Johns Hopkins University, 1995.

15 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995. doi:10.1145/200836.200853.

16 Yi-Jun Chang and Da Wei Zheng. Fully scalable massively parallel algorithms for embedded planar graphs. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 4410–4450, 2024. doi:10.1137/1.9781611977912.155.

17 Xi Chen, Vincent Cohen-Addad, Rajesh Jayaram, Amit Levi, and Erik Waingarten. Streaming Euclidean MST to a constant factor. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 156–169, 2023. doi:10.1145/3564246.3585168.

18 Vincent Cohen-Addad, Vahab Mirrokni, and Peilin Zhong. Massively parallel k-means clustering for perturbation resilient instances. In *Proceedings of the 39th International Conference on Machine Learning (ICML 2022)*, pages 4180–4201, 2022. URL: https://proceedings.mlr.press/v162/cohen-addad22b.html.

19 Richard Cole and Michael T. Goodrich. Optimal parallel algorithms for point-set and polygon problems. *Algorithmica*, 7(1):3–23, 1992. doi:10.1007/BF01758749.

20 Sam Coy and Artur Czumaj. Deterministic massively parallel connectivity. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC 2022)*, pages 162–175, 2022. doi:10.1145/3519935.3520055.

21 Artur Czumaj, Guichen Gao, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Pavel Veselỳ. Fully-scalable MPC algorithms for clustering in high dimension. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, page 50, 2024.

22 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. doi:10.1145/1327452.1327492.

23 Michael Dinitz and Yasamin Nazari. Massively parallel approximate distance sketches. In *Proceedings of the 23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*, pages 35:1–35:17, 2019. doi:10.4230/LIPIcs.OPODIS.2019.35.

24 Michal Dory, Orr Fischer, Seri Khoury, and Dean Leitersdorf. Constant-round spanners and shortest paths in congested clique and MPC. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC 2021)*, pages 223–233, 2021. doi:10.1145/3465084.3467928.

25 Alessandro Epasto, Mohammad Mahdian, Vahab Mirrokni, and Peilin Zhong. Massively parallel and dynamic algorithms for minimum size clustering. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 1613–1660, 2022. doi:10.1137/1.9781611977073.66.

26 Arnold Filtser. Scattering and sparse partitions, and their applications. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 47:1–47:20, 2020. doi:10.4230/LIPIcs.ICALP.2020.47.

27 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 129–138, 2018. doi:10.1145/3212734.3212743.

28 Michael T Goodrich. Randomized fully-scalable bsp techniques for multi-searching and convex hull construction. In *Proceedings of the 8th annual ACM-SIAM symposium on Discrete algorithms (SODA 1997)*, pages 767–776, 1997.

29 Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999. doi:10.1137/S0097539795294141.

30 Michael T. Goodrich and Nodari Sitchinava. Parallel algorithms in geometry. In *Handbook of Discrete and Computational Geometry*, pages 1225–1239. Chapman and Hall/CRC, 2017.

31 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the Mapreduce framework. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC 2011)*, pages 374–383, 2011. doi:10.1007/978-3-642-25591-5_39.

32 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS 2003)*, pages 534–543, 2003. doi:10.1109/SFCS.2003.1238226.

33 Sariel Har-Peled. *Geometric approximation algorithms*. American Mathematical Soc., 2011.

34 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

35 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, 2007. doi:10.1145/1272996.1273005.

36 Rajesh Jayaram, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. Massively parallel algorithms for high-dimensional Euclidean minimum spanning tree. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 3960–3996, 2024. doi:10.1137/1.9781611977912.139.

37 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for Mapreduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.

38 Nathaniel Lahn, Sharath Raghvendra, and Kaiyi Zhang. A combinatorial algorithm for approximating the optimal transport in the parallel and MPC settings. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, pages 21675–21686, 2023.

39 Abhinandan Nath, Kyle Fox, Pankaj K Agarwal, and Kamesh Munagala. Massively parallel algorithms for computing tin dems and contour trees for large terrains. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2016. doi:10.1145/2996913.2996952.

40 Krzysztof Nowicki and Krzysztof Onak. Dynamic graph algorithms with batch updates in the massively parallel computation model. In *Proceedings of the 32th ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2939–2958, 2021. doi:10.1137/1.9781611976465.175.

41 John H. Reif and Sandeep Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM Journal on Computing*, 21(3):466–485, 1992. doi:10.1137/0221031.

42 Tom White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

43 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2010)*, pages 10:1–10:7, 2010.