# Geometric Spanner in MPC Model

Mohammad Ali Abam* Mohammad Reza Bahrami† Peyman Jabbarzade Ganje‡

**Abstract**

The importance of processing large-scale data is growing rapidly in contemporary computation. In order to design and analyze practical distributed algorithms, recently, the MPC model has been introduced as a theoretical framework. In this paper, we present the geometric spanner construction in the Massively Parallel Computing (MPC) model. Constructing $\theta$-graph using the given $\epsilon$, we modified distributed range tree to find the nearest point to the apex of a $\theta$-cone efficiently and form a $(1+\epsilon)$-spanner in $\mathcal{O}(1)$ rounds and $\widetilde{\mathcal{O}}(S)$ time, where $S$ is the memory size of a single machine.

## 1 Introduction

We consider the problem of finding a geometric spanner for a set of points in $\mathbb{R}^d$ in the MPC model. In this section we talk about both the spanner and MPC model and also we motivate our work. Then the sketch of the proof is presented.

### 1.1 Spanners

Given an edge-weighted graph $G = (V, E)$, the $t$-spanner of $G$ is a weighted graph $S = (V, E')$ where $E' \subseteq E$ and for every two vertices $u$ and $v$, $d_S(u, v) \leq t \cdot d_G(u, v)$, where $d_S(u, v)$ indicates the shortest path distance between $u$ and $v$ in graph $S$. That is, the spanner of graph $G$, is a graph which preserve the distances by a $t$ multiplicative factor. For a fixed graph $G$, the smallest $t$, which preserve spanner property is called dilation or stretch factor.

Among general graphs, we are focused on Euclidean geometric graphs, which are complete graphs with a Euclidean metric edge weight function. Hence, a geometric spanner is a complete Euclidean graph, in which for any two points we have the weight of edge $(u, v) \leq |uv|$, where $|uv|$ indicates the Euclidean distance between points $u$ and $v$.

Also, we can view a spanner as compression for the main graph. Assuredly, the fewest edges it has, the more compressed it is considered. However, we look forward to keeping the dilation as low as possible. Assuming Erdös girth conjecture, there is a tight bound for general graphs that there is a $(2k-1)-$*spanner* with $\mathcal{O}(n^{1+1/k})$ edges [19].

The extensiveness of spanners' applications like approximating shortest path queries, routing distributed and parallel algorithms, designing work efficient PRAM algorithms [13], and etc. make researchers come up with many ideas and variants to build them efficiently.

### 1.2 MPC Model and Geometric Problems

The importance of processing large scale data-set is becoming more crucial every day. Over past years we have faced exponential growth in the speed of generating new data. Recently, many tools have been developed to process these data practically, among them Map-Reduce model [10], Hadoop [21], Spark [22] and Dryad [16]. Due to the diversity of tools and methods, some general theoretical models were built to conceptualise current and forthcoming tools. One of the most realistic and practical abstract models is *Massively Parallel Computing* or **MPC** for short, which was introduced by Karloff et. al [17], as a computation model for Map-Reduce.

In this model [7], [14], [17], it is assumed that there are $M$ machines, each of which has a memory of size $S$ known as local memory and the given input size is $N$ which distributed arbitrarily. Usually, we assume the memory size is polynomially smaller than the input size, i.e. $S = \widetilde{\mathcal{O}}(N^\alpha)$ where $0 < \alpha < 1$. As the whole data must fit into machines, we consider $M \geq \frac{N}{S}$. Also, it is common to add polylog factor to the number of machines, i.e. $M = \widetilde{\mathcal{O}}(\frac{N}{S}) = \mathcal{O}(N/S\ poly(\log N))$. Also, despite K-machine model [18], there is no point to point connection between machines, so there is a $\widetilde{\mathcal{O}}(S)$ communication limit for each machine.

However, while we consider graph problems, given a graph $G = (V, E)$ the total memory $N$ is considered as the number of graph edges $E$ which can be $\mathcal{O}(n^2)$ where $n = |V|$. But we are interested in settings where local memory is small. Thus, there are three studied regimes in the MPC model for graph problems; Strong superlinear where $S \geq n^{1+c}$ for a constant $c > 0$, near-linear where $S = \widetilde{O}(n)$, and strongly sublinear where $S = n^\gamma$ for a constant $\gamma < 1$.

In geometric problems, especially in geometric graphs, as the weight function is defined implicitly without keeping all edge weights on memory, we only con-

*Department of Computer Science, Sharif University of Technology, Tehran, Iran abam@sharif.edu

†Department of Computer Science, Sharif University of Technology, Tehran, Iran mrbahrami@ce.sharif.edu

‡Department of Computer Science, University of Maryland, College Park, MD, US peymanj@cs.umd.edu

sider in the later regime. Henceforth, from now on we consider $n$ points and $M = \widetilde{\mathcal{O}}(n^{1-\gamma})$ machines where each of them has $S = \widetilde{\mathcal{O}}(n^{\gamma})$ local memory for a constant $\gamma < 1$.

### 1.3 Prior Works

There is an extensive line of research on *spanners* and also *geometric spanners*, especially after Chew [9] introduced them in 1986 but term spanner coined by Peleg and Ullman [20]. In addition to some surveys [11], [15], there is a book completely about geometric spanner [19].

Although there is a classic greedy algorithm for computing geometric spanner [4], there are many works to reduce the running time while preserving useful properties like minimizing maximum degree or not using too many more edges. E.g. using $\theta$-graph [5] a geometric spanner can be built in $\mathcal{O}(n \log n^{d-1})$ time with $\mathcal{O}(n)$ edges.

Assuming a large data set, there are some works on streaming mode like [6], which construct a spanning tree in a single path with $\mathcal{O}(\min(m, kn^{1+1/k}))$ edges. Also there is a recent work for general graph by Ghaffari et.al [8], constructs a $\mathcal{O}(k^{1+o(1)})$-spanner in $\mathcal{O}(\log \log n)$ rounds in strongly sublinear regime with $\mathcal{O}(n^{1+1/k} \cdot \log k)$ many edges. Also Ghodsi et al. provide a constant round algorithm in MapReduce framework to build the geometric spanner in $L_1$ metric in $\mathbb{R}^2$ using $\mathcal{O}(n/\epsilon^2)$ edges [3].

In recent years, also, there is a growing line of research to solve the geometric problems in the distributed model; such as triangulation algorithms in 2D and 3D [23], or finding the nearest neighbor in K-Machine mode [12].

### 1.4 Overview of Method

Our algorithm works as follows, for each cone $\sigma$ we build a distributed data structure (*Next Tree*) to obtain the nearest point inside the cone. Inside each cone, it takes $\mathcal{O}(d)$ rounds to find the point which is the nearest one with respect to its projection on the representative edge. This is constant if we naturally assume $d$ is constant. To obtain constant round complexity for all points, we must re-distribute sub-trees stored on each machine among searching round, while we look for the nearest point image on the representative edge. This leads to an algorithm with constant rounds of computation and $\mathcal{O}(S)$ time.

## 2 Preliminaries

**$\theta$-graph** We use an approach based on $\theta$-graph [1] to build our spanner. Given constant $\epsilon$, we choose $\theta$ such that $\cos 2\theta - \sin 2\theta \geq 1/(1+\epsilon)$. On the other hand, a $\theta$-cone is the intersection of $d$ half-spaces such that the angle of any two rays emanating at the cone's apex and being inside the cone is at most $\theta$. Hence we define the canonical cone set $\mathcal{C}$, as a collection of $\mathcal{O}(1/\theta^{d-1})$ interior disjoint $\theta$-cones, where share apex at the origin. For a $p \in \mathbb{R}^d$, translating each $\sigma \in \mathcal{C}$ to the point $p$, constructs our $\theta$-graph. Choosing one of the edges of each $\sigma \in \mathcal{C}$ as representative edge ($\ell_\theta$) the following lemma from [1] can be stated:

**Lemma 1** *Let $\mathcal{C}$ be a collection of $\theta$-cones, where $\cos 2\theta - \sin 2\theta \geq 1/(1+\epsilon)$. Choosing $\ell_\sigma$ as representative edge for each $\theta$-cone $\sigma$ and connecting each point $p$ to $q \in \sigma(p)$ that its projection on $\ell_\sigma$ minimize the Euclidean distant to $p$ lead us to a $(1+\epsilon)$-spanner.*

**Distributed Range Tree** A $d$-dimensional range tree is a data structure that can answer orthogonal range queries in $\mathcal{O}(\log^{d-1} n)$ time. If $d = 1$, the range tree is a balanced binary search tree. For $d > 1$, the range tree consists of a primary structure in which every node has a range tree on $d-1$ dimension for the point below that node.

Agarwal et al. in [2] show that it is possible to build a range tree in the MPC model efficiently. As the range tree should be stored in a distributed fashion, a primary top tree is built which contains $\mathcal{O}(s^{1/5})$ top nodes of the range tree. Hence for each node, the top of the secondary structure build locally and recursively. The procedure continues by using leaves of the primary structure and build the tree recursively. We will use this structure and the result of the following theorem from [2]:

**Theorem 2** *Given a set $P \subset \mathbb{R}^n$, a range tree on $P$ can be built with size $\mathcal{O}(n \log^{d-1} n)$ in constant rounds and $\mathcal{O}(s \log^d s)$ time which can answer orthogonal queries in $\mathcal{O}(1)$ rounds and $\mathcal{O}(\log^{d-1} n)$ time in MPC model.*

## 3 Next Tree

First, we consider the definition of Next Tree. The Next Tree is an augmented distributed range tree that helps to find the nearest point in a cone to the apex of the query cone. To build this we amend the range tree [2].

As we discussed earlier, we need a range tree to find the subset of points inside the query cone $\sigma$. Also, we need to find the nearest point $q \in \sigma(p)$ to connect it to $p$ in terms of $q$'s projection to one of $\sigma(p)$ representative edges or even any arbitrary line at that half-space. Consider a line in one of the mentioned half-spaces as $\ell_\sigma$, We assume that $q'_\sigma$ the projection of $q$ on $\ell_\sigma$ is stored within $q$.

To build the Next Tree, we add one dimension to each point which contains its rank on $\ell_\sigma$. This rank can be computed easily with a constant round of computation [14]. Afterward, a $(d+1)$-dimension distributed range tree is built open this augmented point set $P_\sigma$, just like

the one appeared in [2]. We only alter the query procedure to find the lowest rank point in the canonical subset. This leads us to the following lemma:

**Lemma 3** *For a point set $P \subset \mathbb{R}^d$, Next Tree can be built in $\mathcal{O}(d)$ rounds, in $\mathcal{O}(s \log^{d+1} n)$ time with size $\mathcal{O}(n \log^d n)$.*

A query to Next tree is an orthogonal cone $\sigma(p)$. The desired response is a point $q$ where $q'_\sigma$, the projection of $q$ on $\ell_\sigma$, is the next projected point on $\ell_\sigma$ and $q \in \sigma(p)$.

**Lemma 4** *In d-dimension Next Tree, the next query can be answered in $\mathcal{O}(1)$ rounds and $\mathcal{O}(\log^{d+1} n)$ time.*

**Proof.** Without loss of generality consider a orthogonal cone $\sigma(p) = (x_1, \ldots, x_d)$ which is open to $(+\infty, +\infty)$. Just like distributed range tree for each query, we must go through the data structure. For each $x_i$, where $i \leq d$, our search in Next Tree is a path with at most $\mathcal{O}(\log n)$ points. Just like lemma 3, we must repeat it for each $d$ level. However, at the $d$+1 level, we take a $\mathcal{O}(\log n)$ path just to find the minimum ranked point as the $(d+1)$th coordinate of $q$ is rank of $q_\sigma$, which takes $\mathcal{O}(1)$ round and $\mathcal{O}(\log^{d+1} n)$ time for a single point. □

## 4 Spanner Construction

Briefly, the procedure of constructing the spanner is as follows, as algorithm 1 specify, for each canonical cone $\sigma$, we transform points and build a Next Tree. Then we find the nearest point to each cone's apex and connect them.

**Algorithm 1** Distributed algorithm to build a $(1+\epsilon)-$*spanner* using $\theta$-graph

**Require:** Point set $P$, $\epsilon$, $n$ nodes distributed arbitrary

1: Compute $\theta$ based on $\epsilon$ and canonical $\theta$ - cones $\mathcal{C}$
2: **for all** $\sigma$ in $\mathcal{C}$ **do**
3: `transform_and_order_points(`$\sigma$`,` $\ell_\sigma$`)`
4: $T_\sigma$ = `build_next_tree(`$\sigma$`)`
5: `find_nearest(`$P$`,` $T_\sigma$`)`
6: **end for**

Recall that for each $1 \leq i \leq n$, $\sigma(p_i)$ is translated of the same canonical cone $\sigma \in C$. Henceforth, we need to project all points into $\ell_\sigma$ and also transform all points to form all $\sigma(p_i)$ orthogonal. After that, we must compute the rank of each projected point on $\ell_\sigma$ which can be done in a constant rounds for all points [14]. Hence the following proposition is inferred:

**Proposition 5** *In algorithm 1, the procedure* `transform_and_order_points` project the points onto $\ell_\sigma$ and rank them in $\mathcal{O}(1)$ rounds and $\widetilde{\mathcal{O}}(S)$ time.

Now we must show how `find_nearest` works in parallel for all points. If we want to find the nearest point one by one it takes $\mathcal{O}(n)$ rounds to build the spanner which is unacceptable in our model.

By lemma 4, it is clear that query one point in the Next Tree requires $\mathcal{O}(1)$ rounds of computation. However, we design an algorithm to find the next point, the adjacent vertex in the desired spanner, for all points in constant rounds. To this end, we must be able to initiate the search procedure locally for all machines. Hence, each machine needs to have the top sub-tree or the master node which contains the top $\mathcal{O}(S^{1/5})$ nodes [2]. This can be done on $\log_{S^{1/5}} n = \mathcal{O}(1)$ rounds of computation using the broadcast tree [14]. Afterward, in each machine $M_i$, $s$ points have been searched and in the next step their search procedure must continue among set $\mathcal{M}_i = \{M_1, \ldots, M_s\}$. As a result, we can imagine some cases in which a machine is overwhelmed by $\Omega(S)$ nodes to resume its search procedure. To resolve this problem, at the end of the first searching round, each machine sends the number of applicant nodes to their destination machine say $M_i$. As each machine sends a number to $M_i$, if the sum of these numbers exceeds $S$, $M_i$ should make a copy of its self and inform nodes about the new destinations. $M_i$ choose these machines randomly. Also it sends the number of required machines, so using the broadcast tree, we can handle the distribution. The following lemma shows us that this procedure (algorithm 2) can be done on constant rounds of computation.

**Lemma 6** *Targeted sub-trees to resume the query procedure to find neighbour of the point subject to a cone $\sigma$ can be redistributed to be able to host all requests at a round, with $\mathcal{O}(1)$ rounds of computation and $\mathcal{O}(S)$ time.*

**Proof.** Notice that, the total number of points is $n$ and also a machine requires another helping machine for every $\mathcal{O}(S)$ applicant point. As $S \geq s = n/M$, where $s$ is the number of points hosted by each machine, the total number of sub-trees to redistribute is less than $M$. This is also an upper bound for redistributing a fixed sub-tree. Again using the broadcast tree to distribute the sub-trees. A machine with exceeded query requests should choose uniformly at random $\max(S, r/S)$ machines, where $r$ is the number of arriving requests. Using the broadcast tree this will end up in the constant round. However, if a machine receives more nodes than its memory size, it should drop extra sub-trees. So, we consider $X_i$ the number of guest sub-trees requested to join $M_i$. As $E[X_i] = 1$ we can use chernoff bound:

$$Pr[X_i \geq \log n] \leq \exp(\frac{-\log^2 n}{3}) \leq \frac{1}{n^2}$$

Using union bound, the probability of no machine re-

ceive more than $\mathcal{O}(\log n)$ nodes is:

$$Pr[X_1 \cup \cdots \cup X_M] \leq \sum_{1}^{M} \frac{1}{n^2} = \frac{M}{n^2} \leq \frac{1}{n}$$

So, with high probability, no machine receives more than $\log n$ nodes. Using the fact that $S = \widetilde{\mathcal{O}}(n^\delta)$, the lemma concludes. □

**Algorithm 2** `find_nearest`$(P, T_\sigma)$

**Require:** Point set $P$ and its projection on $\ell_\sigma$ within, Pre-computed and stored $T_\sigma$, Constant $k = \log_{S^{1/5}} n = \mathcal{O}(1)$

1: Distribute the master node to all machines and set as current searching node $M_i$ on each machine
2: **for** $k$ times **do**
3: Perform the search on $M_i$ locally and find the $\mathcal{M}_i$ machine set to resume the search
4: Inform $\mathcal{M}_i$ members about the coming requests
5: Redistribute the machine $M_i$ if necessary
6: **end for**

Using proposition 5 and lemma 3, we conclude that the first two lines of the for loop in Algorithm 2 requires $\mathcal{O}(1)$ rounds and $\widetilde{\mathcal{O}}(S)$ time. Moreover, by lemmas 4 and 6 the `find_nearest` procedure requires $\mathcal{O}(1)$ rounds and $\mathcal{O}(S)$ time. Assuming $d$ and $\epsilon$ are constants leads to the following theorem:

**Theorem 7** *In the MPC model, a geometric $(1+\epsilon)$-spanner can be built using $\mathcal{O}(1)$ rounds of computation and $\widetilde{\mathcal{O}}(S)$ time.*

## 5 Concluding Remarks

We considered constructing a geometric spanner in the MPC model. Benefiting from $\theta$-graph, we designed an algorithm to build this spanner in constant rounds of computation. However, we can consider the same problem to find a spanner with a smaller final degree. Also, we can consider the problem in dynamic or kinetic models if we consider the possibility of temporal changes in our point, with less work.

## References


[1] M. A. Abam and M. de Berg, "Kinetic spanners in $\mathbb{R}^d$," in *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, ser. SCG '09, Aarhus, Denmark: Association for Computing Machinery, 2009, pp. 43–50.

[2] P. K. Agarwal, K. Fox, K. Munagala, and A. 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*, ser. PODS '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 429–440.

[3] S. Aghamolaei, F. Baharifard, and M. Ghodsi, "Geometric spanners in the MapReduce model," in *Lecture Notes in Computer Science*, Springer International Publishing, 2018, pp. 675–687.

[4] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, "On sparse spanners of weighted graphs," *Discrete Comput. Geom.*, vol. 9, no. 1, pp. 81–100, Dec. 1993.

[5] S. Arya, D. M. Mount, and M. Smid, "Dynamic algorithms for geometric spanners of small diameter: Randomized solutions," *Computational Geometry*, vol. 13, no. 2, pp. 91–107, 1999.

[6] S. Baswana, "Streaming algorithm for graph spanners—single pass and constant processing time per edge," *Information Processing Letters*, vol. 106, no. 3, pp. 110–114, 2008.

[7] P. Beame, P. Koutris, and D. Suciu, "Communication steps for parallel query processing," in *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '13, New York, New York, USA: Association for Computing Machinery, 2013, pp. 273–284.

[8] A. S. Biswas, M. Dory, M. Ghaffari, S. Mitrović, and Y. Nazari, "Massively parallel algorithms for distance approximation and spanners," in *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '21, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 118–128.

[9] L. Chew, Paul, "There are planar graphs almost as good as the complete graph," *Journal of Computer and System Sciences*, vol. 39, no. 2, pp. 205–219, 1989.

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[11] D. Eppstein, "Chapter 9-spanning trees and spanners," *Handbook of Computational Geometry*, pp. 425–461,

[12] R. Fathi, A. R. Molla, and G. Pandurangan, "Efficient distributed algorithms for the k-nearest neighbors problem," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 527–529.

[13] S. Friedrichs and C. Lenzen, "Parallel metric tree embedding based on an algebraic view on moore-bellman-ford," *J. ACM*, vol. 65, no. 6, Nov. 2018.

[14] M. T. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the mapreduce framework," in *Proceedings of the 22nd International Conference on Algorithms and Computation*, ser. ISAAC'11, Yokohama, Japan: Springer-Verlag, 2011, pp. 374–383.

[15] J. Gudmundsson and C. Knauer, "Dilation and detours in geometric networks," *Handbook of Approximation Algorithms and Metaheuristics Chapman & Hall/CRC Computer & Information Science Series*, 2007.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007.

[17] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for mapreduce," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10, Austin, Texas: Society for Industrial and Applied Mathematics, 2010, pp. 938–948.

[18] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson, "Distributed computation of large-scale graph problems," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '15, San Diego, California: Society for Industrial and Applied Mathematics, 2015, pp. 391–410.

[19] G. Narasimhan and M. Smid, *Geometric Spanner Networks*. USA: Cambridge University Press, 2007.

[20] D. Peleg and J. D. Ullman, "An optimal synchronizer for the hypercube," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 77–85.

[21] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10, Boston, MA: USENIX Association, 2010, p. 10.

[23] H. Zhou, H. Wu, S. Xia, M. Jin, and N. Ding, "A distributed triangulation algorithm for wireless sensor networks on 2d and 3d surface," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 1053–1061.