PANDORA: A Parallel Dendrogram Construction Algorithm for Single Linkage Clustering on GPU

ABSTRACT

This paper presents Pandora, a novel parallel algorithm for efficiently constructing dendrograms for single-linkage hierarchical clustering, including Hdbscan*. Traditional dendrogram construction methods from a minimum spanning tree (MST), such as agglomerative or divisive techniques, often fail to efficiently parallelize, especially with skewed dendrograms common in real-world data.

Pandora addresses these challenges through a unique recursive tree contraction method, which simplifies the tree for initial dendrogram construction and then progressively reconstructs the complete dendrogram. This process makes Pandora asymptotically work-optimal, independent of dendrogram skewness. All steps in Pandora are fully parallel and suitable for massively-threaded accelerators such as GPUs.

Our implementation is written in Kokkos, providing support for both CPUs and multi-vendor GPUs (e.g., Nvidia, AMD). The multithreaded version of Pandora is 2.2× faster than the current best-multithreaded implementation, while the GPU Pandora implementation achieved 6-20× on AMD MI250X and 10-37× on Nvidia A100 speed-up over multithreaded Pandora. These advancements lead to up to a 6-fold speedup for Hdbscan* on GPUs over the current best, which only offload MST construction to GPUs and perform multithreaded dendrogram construction.

ACM Reference Format:

. 2024. PANDORA: A Parallel Dendrogram Construction Algorithm for Single Linkage Clustering on GPU. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A dendrogram, a tree-like structure that encapsulates the hierarchical nature of data, is a critical tool in machine learning. Its ability to visually represent clusters' formation, merging, and splitting has applications in diverse fields, from linguistics [11], astronomy [13] and psychometry [47] to document classification [29] and spatial-social network visualization [30]. In the computational and molecular biology world, dendrograms are used for representing phylogenetic trees [28], elucidating gene clustering [21] and the evolutionary relationships among biological taxa [12]. They are instrumental in decoding gene co-expression [32, 46], protein-protein interaction networks, speciation rates, and genetic mutations [34, 49].

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

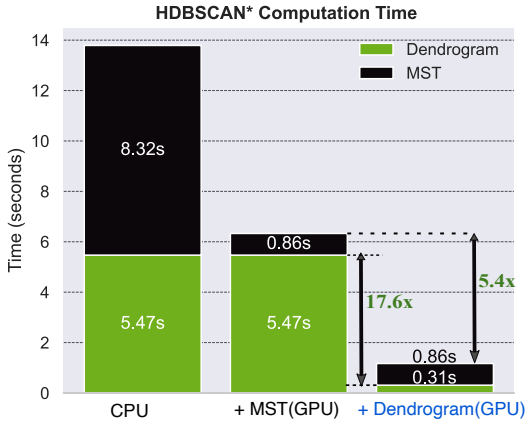


Figure 1: Time taken by Hdbscan* components construction (Euclidean minimum spanning tree (MST) and dendrogram) on AMD EPYC 7A53 CPU and AMD MI250X GPU for *Hacc37M* dataset.

In most hierarchical clustering algorithms, the decision of how to combine or split clusters is done through a use of a distance metric (e.g., Euclidean) [14, 24, 44]. Algorithms differ in their definition of the dissimilarity between two clusters. In this work, we will focus on the *single-linkage clustering*, which defines the distance between two clusters to be the minimum distance among all pairs of points such that the points in a pair do not belong to the same cluster. Our choice is motivated by its use in the popular Hierarchical Density-Based Spatial Clustering of Applications with Noise (Hdbscan*) algorithm [9].

In general, dendrogram construction is considered to be an inexpensive operation. It is often done as step in a larger procedure. For example, in Hdbscan*, constructing MST (as part of Hdbscan*) for high-dimensional data relies on high-dimensional nearest neighbor search, an expensive procedure dwarfing the dendrogram construction cost. Another reason is that the dendrogram construction is often done for the datasets of moderate size.

None of these assumptions hold for the problems we explore in this work. We are focus on the large datasets of the low dimensional data. In this case, dendrogram construction becomes a dominant cost. Figure 1 (middle) shows the status quo for an astronomy dataset *Hacc37M*, where the MST construction is performed on a GPU, and the dendrogram construction is done on CPU. We see that the dendrogram construction takes 86% of the overall time, hampering the overall performance of the Hdbscan* algorithm. The goal of this paper is thus to bridge this gap by presenting a new parallel dendrogram construction algorithm suitable for GPU architectures.

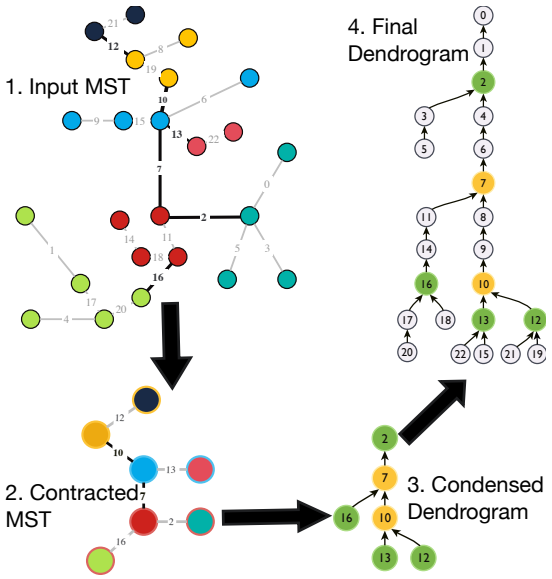


Figure 2: A high-level visualization of the Pandora algorithm. The original MST (top left) is contracted (bottom left). Using the dendrogram corresponding to the contraction (bottom right), the original dendrogram is recovered by edge reinsertion from the same level. The dendrograms are shown using the edges numbering of the original MST; dendrogram leaf nodes, corresponding to the data points, are omitted.

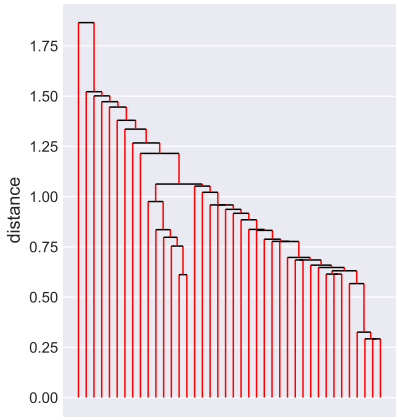


Figure 3: An example of a highly skewed dendrogram constructed from a 40 point sample taken from a 3D Gaussian distribution using HDBSCAN* mutual reachability distance with $minPts = 2$.

We introduce a novel parallel algorithm for constructing single-linkage dendrograms. Our approach is very different from the existing top-down and bottom-up approaches (see Section 2). It effectively handles highly skewed dendrograms such as shown on Figure 3, which are common as we will demonstrate. The algorithm contracts a subset of edges in an MST to yield its coarser version. We efficiently compute the dendrogram for this coarse MST and reconstruct the dendrogram for the original MST by reintegrating all the previously contracted edges. The contraction is applied recursively to ensure the work optimality of our algorithm. This process is illustrated in Figure 2 using an MST with two levels of contraction. Figure 1 (right) shows that we are able to achieve our goal, improving the construction time by 17×, now taking 26% of the overall time for this dataset.

Our proposed parallel dendrogram construction algorithm is work-optimal, highly parallel, and efficient, even when dealing with highly skewed dendrograms. All the steps are highly parallelizable and can be easily adapted for multicore CPUs and GPU architectures using parallel constructs such as parallel loops, reductions and prefix sums. We implemented our algorithm using the performance-portable Kokkos library [41], which marks the first known GPU implementation for dendrogram construction.

We evaluate our algorithm on various real-world and artificial datasets, and compare it to the best-known open-source multi-core CPU implementation. Our experimental results reveal that our multi-core CPU implementation is twice as fast as the state-of-the-art, while the GPU variant achieves 15-40× speedup compared to the multi-core CPU performance.

Our work significantly advances the state-of-the-art in parallel HDBSCAN* clustering computation on GPUs for large datasets. Combined with recent developments in parallel Euclidean MST computation on GPUs [36], our algorithm facilitates rapid HDBSCAN* clustering computation on modern hardware for low dimensional datasets. For example, a single Nvidia A100 GPU can now compute HDBSCAN* clustering in under one second for a 37M cosmological problem, and around six seconds for a 300M uniformly distributed point cloud.

2 BACKGROUND

2.1 Hierarchical clustering

Hierarchical clustering is a method of clustering data that creates a hierarchy of clusters, known as a dendrogram. Unlike other methods, it does not require the number of clusters in advance. THierarchical clustering comes in two flavors: agglomerative, which merges clusters from the bottom up, and divisive, which splits clusters from the top down.

Distance is key in hierarchical clustering, influencing how data points are considered similar or dissimilar. This leads to various hierarchical clustering types, such as single linkage, complete linkage, and average linkage, each with a different distance calculation method.

2.2 Single linkage clustering

Single linkage clustering [16], or nearest neighbor clustering, measures the distance between clusters by the shortest distance between any two points in the clusters. This method excels at finding clusters with irregular shapes, unlike complete linkage and average linkage, which use the longest and average pairwise distances, respectively.

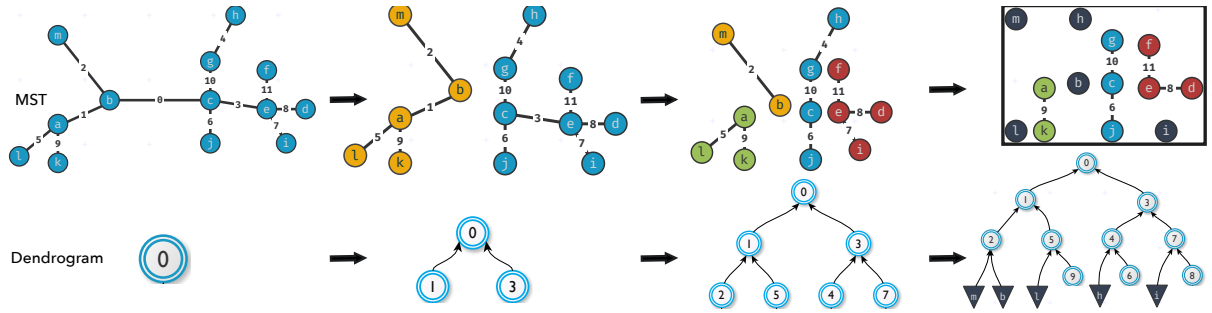


Figure 4: Steps in top-down dendrogram construction(Section 2.5.1). The dendrogram starts with the heaviest edge in MST as the root, which is removed to divide the tree into two connected components. In subsequent steps, the heaviest edge among all components is identified, and their parent is the heaviest edge from the previous step. This process is repeated recursively for each component of the tree.

Table 1: Available open-source dendrogram construction implementations

Implementation	Description
`scikit-learn` (Python) [33]	Sequential implementation
`hdbscan` (Python) [31]	Sequential implementation
`hdbscan` (R) [18]	Sequential implementation
Wang *et al.* [43]	Multi-threaded implementation in shared memory
`rapidsai` [22]	Parallel MST implementation using GPUs, sequential dendrogram construction

Single linkage clustering (SLC) is a versatile technique with applications in bioinformatics, astronomy, and image processing. In bioinformatics, SLC is used to analyze gene expression data [38]. Astronomers apply a similar approach, the Friends-of-friends algorithm, to identify galaxy clusters [13]. In image processing, SLC aids in grouping similar pixels for image segmentation [45] and examines image morphology using tree structures [8, 19, 39]. Additionally, clustering methods based on the Minimum Spanning Tree (MST), which are related to SLC, are used in various studies [23, 27, 46, 48].

Single linkage clustering methods, such as HDBSCAN*, begin by creating a minimum spanning tree (MST). This requires a distance matrix that holds the pairwise distances between data points. However, in spatial point clustering, the matrix isn't explicitly formed. Instead, spatial search trees like kd-trees are used [4, 36, 43]. To compute the MST, algorithms like Prim's, Kruskal's, or Borůvka's can be employed [7, 26, 35].

The second step in single linkage clustering is to construct a dendrogram that reveals a hierarchical cluster structure. The dendrogram is produced using the MST, which is explained in greater detail.

2.3 HDBSCAN*

HDBSCAN* (or hierarchical DBSCAN) [9], a variant of single-linkage clustering, is a hierarchical, density-based algorithm that groups points into clusters based on local density compared to their larger neighborhood. It is adept at finding clusters of various shapes and densities, with the primary parameter being *minPts*, which estimates local density. HDBSCAN* employs the so called *mutual reachability distance*, an adaptation of the Euclidean distance that incorporates local density.

Traditional DBSCAN requires a user-defined epsilon (ϵ) parameter to determine the maximum distance between points in a neighborhood or cluster. HDBSCAN*, on the other hand, eliminates the need for a manually set ϵ value by using a dendrogram that contains clustering information for a range of epsilon values.

Table 1 presents a list of existing HDBSCAN* implementations. Previous research on HDBSCAN* parallelization mainly concentrated on optimizing distance calculations and MST computations. This includes the single-linkage clustering in the cuML library, which employs RAPIDS.ai/Raft [22]. Only one study [43], to our knowledge, explores the parallelization of dendrogram construction for HDBSCAN* on multithreaded platforms, which we will discuss in greater detail.

2.4 Dendrogram

Dendrogram: A dendrogram is a directed tree structure used to represent hierarchical clustering. The tree consists of leaf nodes that represent data points and internal nodes that represent clusters of data points. The relationships between clusters are conveyed through directed edges, indicating whether a cluster contains or is contained in another cluster.

Dendrogram in Single-Linkage Clustering: Single-linkage clustering methods use the edges of the MST to represent the internal nodes of the dendrogram. If an edge is removed from the MST, it indicates splitting a cluster into two smaller clusters. As a result, clusters correspond to MST edges, and removing them leads to cluster separation. Removing an edge can only divide a cluster into two, hence the dendrogram is typically a binary tree.

2.5 Dendrogram construction algorithms

2.5.1 ***Top-down dendrogram construction.*** The top-down approach for constructing a dendrogram from a given Minimum Spanning Tree (MST) employs a divide-and-conquer strategy. This method hinges on the principle that the dendrogram's root corresponds to the largest edge in the MST. Removing this edge divides the tree into two subtrees, which may be as small as a single vertex. These subtrees form the child nodes of the removed edge in the complete dendrogram. The process is then recursively applied to

Algorithm 1 Top-down dendrogram construction for a given MST $T = (V, E)$.

```
1:  function DendrogramTopDown(T)
2:      if |V| = 1 then
3:          D ← {V}
4:      else
5:          D = ∅
6:          e ← the largest edge in E
7:          Remove e from E, splitting T into T_1 and T_2
8:          for all T_i ∈ {T_1, T_2} do
9:              D_i ← DendrogramTopDown(T_i)
10:             Set root note of D_i as a child of e in D
11:     return D
```

each subtree, removing the largest edge until all edges are eliminated. First three steps for an example MST is shown in Figure 4. Pseudo-code for this method is presented in Algorithm 1.

In some specific cases, such as in image morphological studie [6, 20], a variant of this approach has shown good parallel performance. However, for general dendrogram construction, this approach has several drawbacks.

The top-down approach is particularly prone to underperforming with skewed dendrograms, which are common in real-world data. For optimal performance, the algorithm must split the tree by removing the largest edge, aiming for two subtrees of equal size. However, in skewed scenarios, this can result in highly disproportionate subtree sizes, occasionally reducing one subtree to a single vertex. This results in:

- *Increased asymptotic cost:* The cost of the algorithm is $O(nh)$, with h representing the dendrogram's height. In the case of skewed dendrograms, this cost surpasses the $O(n \log n)$ cost associated with well-balanced dendrograms. Therefore, the top-down algorithm is not work-optimal for highly skewed dendrograms.
- *Limited parallelism:* The imbalance of two subtrees after an edge removal limits the available parallelism. Moreover, the computational depth (the number of required parallel steps) is $O(h)$, much higher than the ideal $O(\log n)$.

2.5.2 Bottom-up dendrogram construction. The bottom-up processes the edges in order from the smallest to the largest. For each edge, it identifies the clusters containing its vertices, and creates a new parent cluster by merging the vertices' clusters. To keep track of the cluster membership, it utilizes the union-find structure [40]. Algorithm 2 shows pseudo-code for this approach.

In contrast to the top-down algorithm, the bottom-up approach is work-optimal for any dataset. The most demanding operation, sorting, has $O(n \log n)$ complexity. Processing edges has an asymptotic cost $O(n\mathcal{A}(n))$, where $\mathcal{A}(n)$ represents the inverse Ackerman function [40]. This gives the overall worst-case time complexity of $O(n \log n)$.

The main drawback of the algorithm is that the edges can only be processed sequentially. For a given edge, it is impossible to say when it should be processed given the information only about its

Algorithm 2 Bottom-up dendrogram construction using union-find for a given an MST $T = (V, E)$. E is assumed to be sorted, $E = \{e_i\}$.

```
1:  function DendrogramBottomUp(T)
2:      Initialize an empty union-find structure UF
3:      Initialize set R with invalid entries ◊
4:      for i = 1 to |E| do
5:          Let v_1, v_2 be the vertices of e_i
6:          for k = 1, 2 do
7:              v̄_k ← UF.find(v_k)
8:              if r_{v̄_k} ≠ ◊ then
9:                  Set e_i to be the parent for r_{v̄_k}
10:             else
11:                 Set e_i to be the parent for v_k
12:         UF.union(v_1, v_2)
13:         r_{UF.find(v_1)} ← i
```

vertices or adjacent edges. This is due to the non-local nature of the dendrogram, where the parents of an edge may come from a completely different part of the graph. Thus, standard methods to parallelize sequential algorithms, such as [5], cannot be used here.

2.5.3 Mixed dendrogram construction. Wang et al. [43] combined top-down and bottom-up approaches to create a parallel algorithm for the shared memory architecture. The algorithm avoids the limitations of the sequential bottom-up approach by first removing a set of the largest edges (a tenth or a half) in a top-down fashion, splitting the tree into several subtrees. The dendrograms for the subtrees and the top tree are constructed using the bottom-up approach, then stitched together.

The algorithm exhibits higher degree of parallelism compared to the sequential counterpart. However, it is still subject to the same limitations for constructing highly skewed dendrograms, leading to work inefficiency and imbalance, particularly problematic on GPUs. Moreover, it relies on the Euler tour implementation for partitioning. Euler tour construction heavily depends on parallel list-ranking, which significantly underperforms on GPUs compared to prefix-sum or sort algorithms.

3 PANDORA: PARALLEL DENDROGRAM COMPUTATION USING TREE CONTRACTION

Pandora leverages dendrogram chains—continuous segments without branching, present in highly skewed dendrograms. Notably, within a chain, the edges are organized by their index. Consider an inverted Y-shaped dendrogram (Fig. 5): it consists of three chains—top, bottom-left, and bottom-right. By assigning each edge to its respective chain, we can efficiently sort and link the chains to reconstruct the full dendrogram.

To identify these chains, we employ a tree contraction method, which generates a condensed version of the original dendrogram.

Algorithm 3 Dendrogram Computation using Tree Contraction

Require: $T = (V, E)$: minimum spanning tree
1: $E_\alpha \leftarrow$ Find edges of contracted tree
2: $T_\alpha \leftarrow$ Construct contracted tree by contracting edges in $E - E_\alpha$ in T
3: $P_\alpha \leftarrow$ Compute dendrogram of contracted tree T_α
4: ▷ *Construct complete dendrogram P from P_α*
5: **for each edge** e **in** $E - E_\alpha$ **in parallel do**
6: ▷ *Find the chain of e using contracted dendrogram P_α*
7: $P_\alpha(e) \leftarrow$ Find parent of e in P_α
8: ▷ *Map e to its corresponding chains in P*
9: $C \leftarrow$ Determine of chain containing e
10: Add e to set of edges in the chain C
11: ▷ *Order and connect chains to form P*
12: **for** each chain C **do**
13: Sort edges in C by their index in E
14: **for** each edge e in C, excluding the first **do**
15: $P(e) \leftarrow$ Find predecessor of e in C
16: **if** e_s is first edge in sorted chain **then**
17: $P(e_s) \leftarrow \alpha$-edge for chain C
18: Connect chains to form the complete dendrogram P
19: **return** P

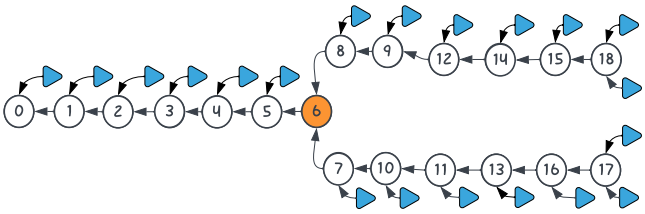


Figure 5: The Pandora leverages dendrogram chains to construct them efficiently. This dendrogram can be divided into three chains: top, bottom-left, and bottom-right.

It condenses each dendrogram chain into a single edge in the contracted dendrogram, allowing us to map all edges to a dendrogram chain and construct the complete dendrogram.

Pandora operates in two main stages. The first is the recursive tree contraction (Section 3.2), where we strategically reduce the tree's size by contracting specific edges. We apply this contraction recursively to determine the dendrogram of the reduced tree. The second stage is the dendrogram expansion (Section 3.3), which involves piecing together the full dendrogram, starting from the most contracted state and incrementally expanding it.

We begin by defining essential terms and notations for our algorithm's description.

3.1 Terminology and notation

3.1.1 Minimum spanning tree structure. Consider a Minimum Spanning Tree (MST) $T = \{V, E, W\}$, where we aim to calculate its dendrogram. Let n_v denote the number of vertices, and $n = n_v - 1$ represent the number of edges in the MST. The dendrogram computation begins with sorting the edges in T by weight in descending order, which requires $O(n \log n)$ time. This sorting step is crucial for the dendrogram's computation, ensuring that edges with equal weights are ordered consistently to preserve the dendrogram's uniqueness and facilitate the validation of our method. For subsequent discussions, we will assume that the edges are already sorted in this manner. We use the following notation to describe the incidence structure of the tree.

Incident edges: For a vertex $v \in V$, the set $\mathsf{Incident}(v)$ includes all edges incident to v. For example, as shown in Figure 7a, $\mathsf{Incident}(a)$ comprises the edges $\{e_0, e_2, e_3, e_5\}$.

Maximum incident edge: $\mathsf{maxIncident}(v)$ denotes the edge with the highest index in $\mathsf{Incident}(v)$. From the previous example, $\mathsf{maxIncident}(a) = e_5$.

Neighboring edges $\mathcal{N}(e)$: For any edge e, the set $\mathcal{N}(e)$ consists of edges that share a vertex with e. Specifically, if e connects vertices v and u, then $\mathcal{N}(e) = \mathsf{Incident}(v) \cup \mathsf{Incident}(u)$.

Edge contraction of a tree We can create a contracted tree $T_c = (V_c, E_c)$ from a tree T and a subset of edges E_c. To do this, we contract the edges in the set $E - E_c$. Initially, V_c is identical to V. For each edge $e = (u, v) \in E - E_c$, we merge u and v into a single supervertex vu, removing u and v from and adding supervertex vu to V_c. The supervertex vu inherits the neighbors of u and v, except for u and v themselves. This contraction is repeated for all edges in $E - E_c$. The resulting contracted tree T_c comprises the modified vertex set V_c and the edge subset E_c.

3.1.2 Dendrogram structure. A dendrogram is a directed rooted binary tree, denoted as $\mathcal{D} = \{V_d, E_d\}$. Its vertex set V_d comprises two types of nodes: vertex nodes, representing the vertices of the Minimum Spanning Tree (MST), and edge nodes, representing the MST's edges. Thus, we have $V_d = V \cup E$, with vertex nodes located at the leaves corresponding to individual data points, and edge nodes as internal nodes signifying clusters.

The dendrogram's structure is established through directed links that outline parent-child relationships between nodes. These relationships determine the edge set E_d, as defined by the parent function P. Specifically, E_d is composed of directed edges $(v \rightarrow u)$ where $P(v) = u$, with v being a member of V_d—either a vertex or an edge of the MST—and u representing an edge in the MST. Thus, dendrogram computation is equivalent to determining the parent P for all nodes in V_d.

Parent of a vertex-node: In a dendrogram, the parent of a vertex-node $v \in V$ is the edge that disconnects v from the tree when removed during the top-down process. This process entails sequentially eliminating edges in $\mathsf{Incident}(v)$, beginning with the heaviest (the smallest index) and concluding with the lightest (the largest index). Thus, the dendrogram parent of vertex v is the edge in $\mathsf{Incident}(v)$ with the largest index.

$$P(v) = \mathsf{maxIncident}(v) \qquad \forall v \in V; \tag{1}$$

For example, in Figure 7a, $P(a) = e_5$. The incidence structure of the tree allows us to determine the parents of all vertex nodes $v \in V$. However, *identifying the parents of the edge nodes* presents the main challenge.

Types of edge nodes: We can classify edge nodes in a dendrogram based on the number of vertex nodes they have as their children. In a binary dendrogram, each edge node has exactly two children,

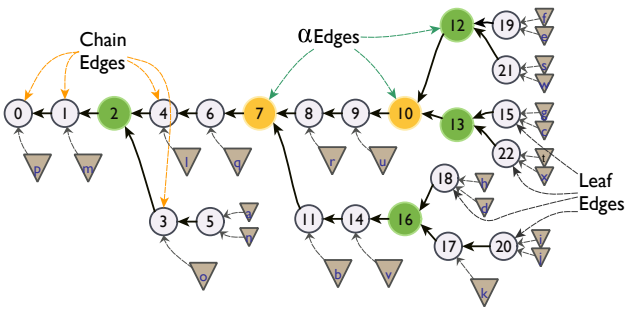


Figure 6: Dendrogram corresponding to the MST in Figure 7a. We mark the vertex nodes as triangles and edge nodes as circles. The edge nodes are further classified into leaf, chain, and α-edges.

which can be either an edge or a vertex node. This leads to three types of edge nodes (shown in Figure 6):

- **Leaf edges:** have two vertex nodes as children.
- **Chain edges:** have one vertex node and one edge node as children.
- **α-Edges:** do not have any vertex node as a child; both of their children are edge nodes as well.

Using the MST's local incidence structure and Equation (1), we can identify the parent of a vertex node. This information allows us to calculate the number of children for any edge node. Consequently, we can classify the edge node as a leaf, chain, or α-edge based on its local incidence structure alone.

However, discerning the parent of an edge node through this local structure alone is challenging due to the more complex parent-child relationships between edge nodes, which often extend beyond immediate neighbors in the MST. For example, in Figure 7b, edge node e_2 has e_1 as its parent, yet e_1 and e_2 are situated in completely separate sections of the tree.

3.1.3 Dendrogram chains and skewness. **Dendrogram chains:**A dendrogram 'chain' is a lineage in a dendrogram that extends without branching. It comprises a series of chain edges followed by a final non-chain edge, which can be either a leaf or an α edge. Each edge node in the chain, except for the last one, has a single child that is the subsequent edge in the chain. The chain's end is marked by a leaf or an α edge. Chains ending in a leaf edge are referred to as *leaf chains*.

Skewness of the Dendrogram: We define a dendrogram's skewness as the ratio of the height of the dendrogram to its ideal height = $\log_2 n$. A large number of chains in the dendrogram can lead to an increase in its height and consequently, its skewness.

Developing a parallel dendrogram algorithm is difficult because real-world dendrograms are often highly-skewed. Even dendrograms constructed from low dimensional Gaussian distributions have heights far from a balanced tree height. This is a common occurrence, as we demonstrate in our results section for various datasets, from GPS location data to cosmology and power usage (see Table 2.)

3.2 Recursive tree contraction

Pandora constructs a condensed version of MST by contracting all edges except the α edges. The dendrogram of this condensed MST is identical to the one obtained by merging chain and leaf nodes in the full dendrogram. This simplified dendrogram effectively captures the full dendrogram's structure.

Computing α-Edges: An α-edge is a type of edge-node that has two children that are also edge-nodes. If an edge-node $e_k = \{v, u\}$ has a vertex node as a child, it will be either v or u. The parent of v is given by $P(v) = \text{maxIncident}(v)$. In case k is not equal to $\text{maxIncident}(v)$ and $\text{maxIncident}(u)$, then e_k is not a parent of either vertex node incident on it. This means that both its children are edge-nodes. Therefore, an edge-node $e_k = \{v, u\}$ is an α-edge if:

$$k \neq \text{maxIncident}(v) \text{ and } k \neq \text{maxIncident}(u). \quad (2)$$

Equation (2) allows for the identification of all α-edges using a constant-time operation for each edge.

In Figure 7, we demonstrate the process. Let's look at the Minimum Spanning Tree (MST) example in Figure 7a. In Figure 7b, we highlight the α-edges of the MST. For example, $e_{16} = \{i, d\}$ is an α-edge because $\text{maxIncident}(i) = 20$ and $\text{maxIncident}(d) = 18$, which are both different from 16. None of the terminal edges are α-edges. For instance, $e_1 = \{m, k\}$ is not an α-edge because $\text{maxIncident}(m) = 1$. Additionally, several internal edges like e_{20} and e_{17} are also non-α edges.

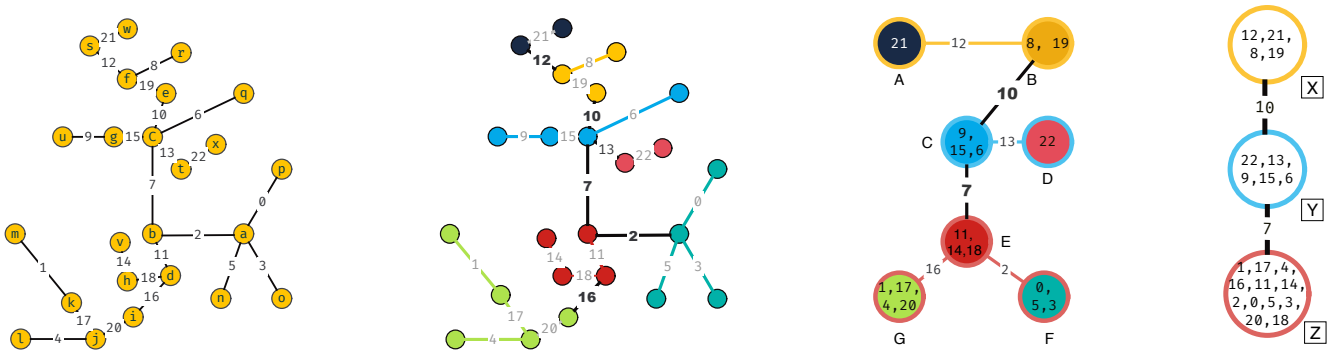
Computing α-MST: We first identify the α edges in the original tree. Then, we contract the remaining non-α edges to create a new tree called α-MST (T_α). In T_α, each vertex is an α-vertex, representing multiple vertices from the original tree that have been contracted. We also keep track of the mapping between the original vertices and their counterparts in T_α, which is important for tracing back to the original structure.

For example, in Figure 7b, we show an MST with highlighted α edges. We contract the non-α edges to obtain the contracted tree shown in Figure 7c. The vertices in Figure 7b that are merged to form a supervertex are colored with the same color. In Figure 7c, vertices a, n, o, and p are merged together to form a single supervertex, all colored cyan.

Multilevel tree contraction: The dendrogram of T_α can be computed by recursively applying the same edge contraction strategy. This leads to a multilevel tree contraction scenario. In each iteration, we find α-edges present at that level of the contracted tree, and contract the remaining edges to get the tree for the next iteration. The recursion stops when there are no more α-edges left. At this point, we get a single chain dendrogram, obtained by sorting the edges according to their indices.

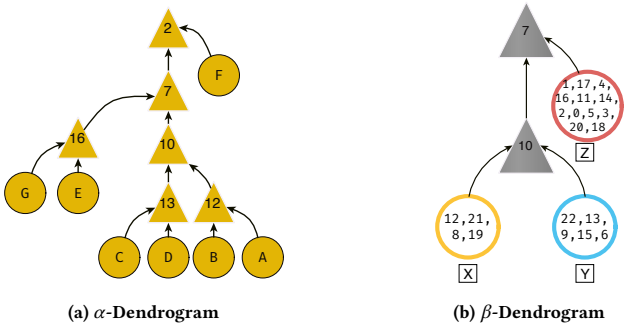
Figure 7d illustrates the β-MST, which emerges from the second contraction level applied to the α-MST depicted in Figure 7c. The β-MST can no longer be contracted, hence the recursion concludes. The final contraction stage is represented by the β-dendrogram, displayed in Figure 8b.

To summarize, we begin with a complete Minimum Spanning Tree (MST). We create a series of smaller trees by performing multilevel tree-contractions on this tree. We continue this process until we have a tree without any α edges. The dendrogram of this tree forms a single chain, which we can obtain by sorting. This results



(a) An MST (not to scale) (b) highligted α-edges (c) α-MST (d) β-MST

Figure 7: Recursive Tree Contraction (Section 3.2). The original MST is shown in Fig. 7a. Fig. 7b highlights the α-edges of the same MST. Removing these edges results in a division of the tree into components, each marked with a different color in Fig. 7b. These components are then contracted into α-vertices. The α-vertices and α-edges form the α-MST, the first level of contraction, depicted in Fig. 7c. This contraction process continues to a second level to form the β-MST, as shown in Fig. 7d. Both α and β MST encapsulate their respective edges within supervertices.



(a) α-Dendrogram (b) β-Dendrogram

Figure 8: The α-dendrogram and β-dendrogram for the α-MST (Figure 7c) and β-MST (Figure 7d) respectively.

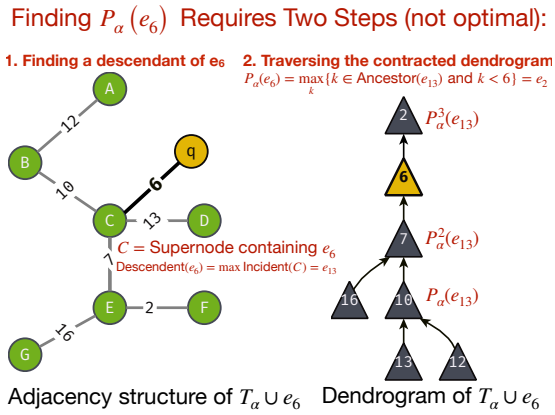


Figure 9: Inserting a non-α edge e_6 into the α-dendrogram. In this process, a single level contraction is done within the supervertex to find the parent of the edge. For example, in the case of edge e_6, we identify its parent by finding the maximum incident edge of the supervertex C, which is e_{13}. We consider e_{13} as a descendant of e_6. To locate the parent of e_6, we go through the dendrogram upwards and select the ancestor with the highest index among all ancestors of $P_\alpha(e_{13})$. This way, we determine that the parent of e_6 is e_2, represented in the dendrogram of $T_\alpha \cup e_6$. However, this accurate method can be inefficient because it may require traversing the entire dendrogram.

in a highly compact dendrogram. In the next section, we explain how to expand this condensed dendrogram into a comprehensive one.

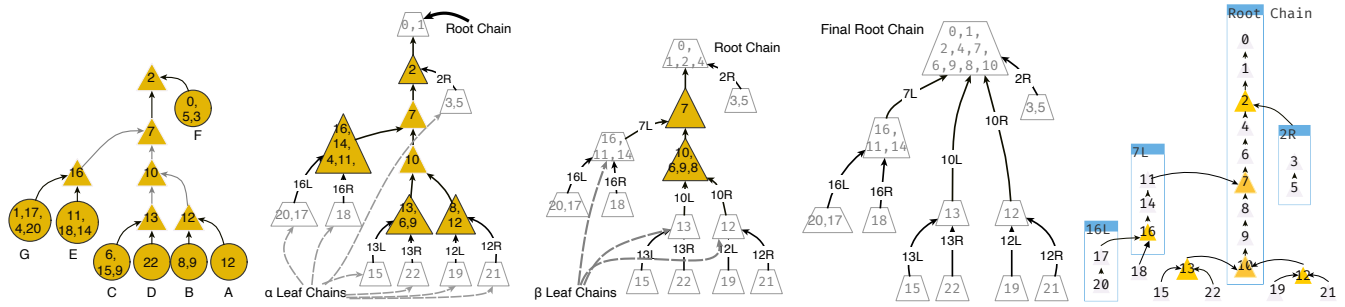
3.3 Efficient dendrogram expansion

In this section, we will explain how to construct a complete dendrogram from a condensed dendrogram, which we call *expansion*. Then, we explain the expansion process for a single-level contraction in Section 3.3.1. However, single-level contraction is not optimal for reconstructing a dendrogram. Therefore, we have developed an expansion algorithm that utilizes all contraction levels, described in Section 3.3.2.

3.3.1 Dendrogram expansion from single-level tree contraction. Given an input Minimum Spanning Tree (MST) called T, a contracted tree containing all the α edges called T_α, and the dendrogram of T_α specified with the parent-child relation P_α, our objective is to assign each non-α edge to a specific dendrogram chain. To accomplish this, we follow these steps:

(1) Find the α-vertex $V_\alpha(e)$ containing e.
(2) Determine the dendrogram parent of $V_\alpha(e)$ in α-dendrogram: $P_\alpha(V_\alpha(e))$.
(3) Traverse the α-dendrogram to find the $P_\alpha(e)$: Starting from $P_\alpha(V_\alpha(e))$. and traverse the dendrogram upwards until an α edge with a smaller index than e is encountered.

Let's consider how to map the edge e_6 into the dendrogram for the minimum spanning tree (MST) shown in Figure 7. The α-vertex that contains e_6 is denoted as $V_\alpha(e_6) = C$ in Figure 7c. The parent of $V_\alpha(e_6)$ in the α-dendrogram is $P_\alpha(V_\alpha(e_6)) = e_{13}$ shown in Figure 8a. To find the parent of e_6, we traverse the alpha dendrogram from bottom to top, starting at e_{13}. We look for an α edge with a lower index than e_6(see Figure 9). In this case, the lower-indexed edge is



(a) α-Dendrogram (b) α-Leaf chains (c) β-Leaf chains (d) Final chains (e) Final Dendrogram

Figure 10: **The process of expanding the complete dendrogram from the contracted dendrogram. 1. In Figure 10a, the α dendrogram is shown, with triangles representing α edges and circles representing α vertices. All non-α edges are displayed within their corresponding α vertices. 2. First, we identify non-α edges that belong to an α leaf chain by comparing it to the parent of the α vertex containing the edge (Figure 10a). Any edge with a higher index than the α parent is considered part of the α leaf chain. 3. We then check the non-α edges that are not part of any leaf chains to see if they belong to a β leaf chain. To do this, we compare each non-α edge to the β edge. The β edge is the parent of the α parent we identified in the previous step. Any edge with a higher index than the β parent is marked as part of its β leaf chain(Figure 10c). This process continues until all edges are assigned to a leaf chain of some level, or there are no more contraction levels remaining. 4. Any unassigned edges are allocated to the root chain if further contraction is not possible, as depicted in Figure 10d. 5. Finally, each chain is sorted to form partial dendrograms. These partial dendrograms are then merged to produce the final dendrogram, as shown in Figure 10e.**

e_2, which becomes the α-parent of e_6. Since e_7 is placed on the left side of e_2, we assign e_6 to the chain $2L$.

However, this method is not optimal as it requires traversing the alpha dendrogram in a bottom-up order for all non-α edges. In the worst case, the height of the alpha dendrogram tree can be $O(n)$. Consequently, finding chains for all non-α edges would require $O(n^2)$ work.

3.3.2 Efficient dendrogram expansion from multilevel tree contraction. We can optimize the dendrogram expansion process by making two key observations.

First, we can quickly identify edges that are part of a leaf chain without traversing the entire α dendrogram. Second, for edges that are not in a leaf chain of the α dendrogram, we can efficiently check if they are in a leaf chain of the β dendrogram.

By recursively applying this process, we can associate all edges with a leaf chain at some level. Instead of traversing the α dendrogram from the bottom up, which can be inefficient due to its height, we start checking for leaf chain membership at level of dendrograms. This approach is more efficient since the number of contraction levels $\log_2 n$.

Leaf Chains: Leaf chains are linked to their respective dendrograms. An α leaf chain refers to a sequence that concludes with a leaf edge in the dendrogram. For example, in Figure 10e, the sequence denoted by $16L$ qualifies as an α leaf chain. Upon removing all α leaf chains from a dendrogram, new leaf chains emerge with an α edge as terminal, termed β leaf chains. A β leaf chain may encompass multiple α chains that are not leaves, and the α edges linked to these chains become part of the β chain. These α and non-α edges together create an unbroken lineage within the full dendrogram. This concept of leaf chains can be extended to higher-level contractions as well.

Mapping edges to a leaf chain: To construct the dendrogram efficiently, we utilize a constant-time method to determine if an edge is part of a leaf chain at any level. We aim to identify the earliest contraction level at which each edge becomes part of a leaf chain.

For each non-α edge e, we first check if it belongs to an α leaf chain by comparing the index of the α parent of $V_\alpha(e)$ to the index of e. If the α parent's index is lower, e is part of an α leaf chain. If not, we check for inclusion in a β leaf chain by examining the β parent of $V_\beta(e)$ in the β dendrogram and comparing it to e. Determining an edge's leaf chain membership at any level takes constant time ($O(1)$).

To map non-α edges to their respective chains, we check if they are part of an α leaf chain. If not, we then check if they belong to a β leaf chain, and so on, until the edge is placed in a chain. Any edges not assigned to a chain at the final level are grouped together in the root chain. The maximum number of contraction levels determines the cost of associating an edge with a leaf chain.

Example: To map a non-α edge, such as e_{15}, to a chain, we first identify its α-vertex, C, as shown in Figure 7c. The α-parent of C, $P_\alpha(C)$, is 13 (Figure 8a). Since 15 is greater than 13, e_{15} is part of the leaf chain associated with e_{13}, specifically the $13R$ chain.

Next, consider edge e_{11}, with α-vertex E and α-parent $P_\alpha(E) = 16$. As 16 is greater than 11, e_{11} is not in the leaf chain. We then determine if it's part of a β-leaf chain. The β-vertex containing e_{11} is X (Figure 7d), with β-parent $P_\beta(X) = 7$ (Figure 8b). Since 11 is greater than 7, e_{11} is indeed in a β-leaf chain.

The mapping process is illustrated in Figure 10. We start with the α-dendrogram (Figure 8a), determine the V_α for all edges (Figure 10a), and identify those in an α leaf chain (Figure 10b). Edges not in an α leaf chain are then checked against β leaf chains (Figure 10c). Finally, edges not in a β leaf chain are assigned to the root chain (Figure 10d).

3.3.3 Final dendrogram construction. In the previous step, we assigned all the edges to a leaf chain or root chain. Now, we will use this information to build the entire dendrogram. This process involves two main steps:

Sorting the Chains: Sorting each chain forms partial dendrograms. In the sorted chain, we assign the parent of each edge to its predecessor in the sorted chain, except for the first edge in the chain, which is handled in the next step.

Stitching the Chains: Each chain is a leaf chain of a contracted edge from a certain level of contraction. The parent of the first edge in the chain is marked as the corresponding edge for that chain. For instance, the α-leaf chain $(17, 20)$ corresponds to the α-edge e_{16} (Figure 10b), while the β-leaf chain $(16, 11, 14)$ corresponds to the β-edge e_7 (Figure 10c). Therefore, the parent of e_{17} in the α-leaf chain $(17, 20)$ is the α-edge e_{16}, and the parent of e_{11} is the β-edge e_7, shown in Figure 10e. By connecting chains in this way, the complete dendrogram is formed shown in Figure 10e.

3.4 Prooof of correctness

This section provides an outline of our algorithm's correctness proof. We will introduce the concept of the Lowest Common Dendrogram Ancestor (LCDA), which helps identify the earliest shared ancestor in the dendrogram for any two edges. We will demonstrate that the LCDA of two edges is the edge with the smallest index on their connecting path in the tree. We define a contraction called cluster hierarchy preserving edge contraction, which retains the dendrogram ancestry properties of the original tree. We will outline the conditions under which this contraction preserves the cluster hierarchy, which is related to the presence of LCDAs. Additionally, we will show that contracting α-edges satisfies the necessary conditions for generating accurate dendrograms. The complete proof will be presented elsewhere.

3.5 Asymptotic analysis

We can show that Pandora algorithm can construct the dendrogram of a minimum spanning tree (MST) with n edges in $O(n \log n)$ operations. Again, the full proof appears elsewhere, but here's an outline. This process involves establishing bounds on the number of edges at each level of the contracted MST, including leaf edges (n_l), chain edges (n_c), and α edges (n_α). We demonstrate that the number of α-edges, n_α, is at most $(n-1)/2$ and that the maximum number of contraction levels is $\lceil \log_2(n+1) \rceil$. By using these relationships, we show that the edge contraction cost is $O(n)$ and the dendrogram expansion cost is $O(n \log n)$. The algorithm requires two sorting operations, each costing $O(n \log n)$, before and after the tree contraction. Thus, the overall cost of Pandora is $O(n \log n)$, making it work-optimal.

4 PERFORMANCE PORTABLE IMPLEMENTATION

4.1 Kokkos

In our work, we utilized Kokkos [41], a performance-portable programming model. This model enables code to run across a range of CPU and GPU platforms without additional modifications. Kokkos provides C++ abstractions and supports various hardware through backends, including CPUs and Nvidia, AMD, and Intel GPUs. The library introduces abstractions for execution and memory resources, referred to as "execution space" and "memory space". However, it is important to note that Kokkos does not perform hidden data copies, so users must ensure data accessibility between memory and execution spaces.

Kokkos offers parallel execution patterns, such as parallel for loops, reductions, and scans, effectively abstracting hardware complexities. Unlike CUDA, Kokkos does not require explicit mapping of computation to threads or thread blocks. Instead, it employs internal heuristics to map a kernel to the underlying hardware architecture.

Table 2: Datasets used in experiments

Name	Dim	n_{pts}	Imb†	Ref.†	Desc.†
Ngsimlocation3	2	6M	1e3	[1]	GPS loc
RoadNetwork3	2	400K	150	[25]	Road network
Pamap2	4	3.8M	6e3	[37]	Activity monitoring
Farm	5	3.6M	5e4	[2]	VZ-features[42]
Household	7	2.0M	1e3	[3]	Household power
Hacc37M	3	37M	1e5	[17]	Cosmology
Hacc497M	3	497M	6e5	[17]	Cosmology
VisualVar10M2D	2	10M	3e3	[15]	GAN
VisualVar10M3D	3	10M	1e4	[15]	GAN
VisualSim10M5D	5	10M	43	[15]	GAN
Normal100M2D	2	100M	1e5	-	Random (normal)
Normal300M2D	2	300M	4e5	-	Random (normal)
Normal100M3D	3	100M	4e5	-	Random (normal)
Uniform100M2D	2	100M	1e5	-	Random (uniform)
Uniform100M3D	3	100M	4e5	-	Random (uniform)

†Imb. = Dendrogram imbalance, Ref. = Reference, Desc. = Description

4.2 PANDORA implementation

In the context of the Pandora algorithm, the two main computational tasks are tree contraction and dendrogram expansion. Each tree contraction corresponds to a prefix sum operation. During dendrogram expansion, we map all the contracted edges to their respective chains in parallel. The cost of mapping a contracted edge to a chain is $O(\log n)$, thus mapping all edges in parallel does not result in noticeable load imbalance in any architecture. Lastly, Pandora performs two parallel sort operations: one before contraction and another after mapping all edges to a chain.

The algorithms described in the text use kernels composed of parallel loops, reductions, prefix sums, and sorting operations. Outside of Kokkos, these operations are widely available in many parallel libraries, such as Thrust[10].

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

5.1.1 Competing Implementation. We evaluate Pandora's performance on multithreaded AMD EPYC 7A53, Nvidia A100 and AMD MI250X (single GCD). Our baseline is UnionFind-MT from https://github.com/wangyiqiu/hdbscan [43]. Note that UnionFind-MT involves a parallel multithreaded sort and a sequential Union-find step phase. We used this implementation for verification of our results. To our knowledge, this is the fastest implementation of dendrogram computation available, and we use it as a baseline for comparison. In our implementation, we used Kokkos [41] (version 3.7) for implementing our parallel dendrogram algorithm.

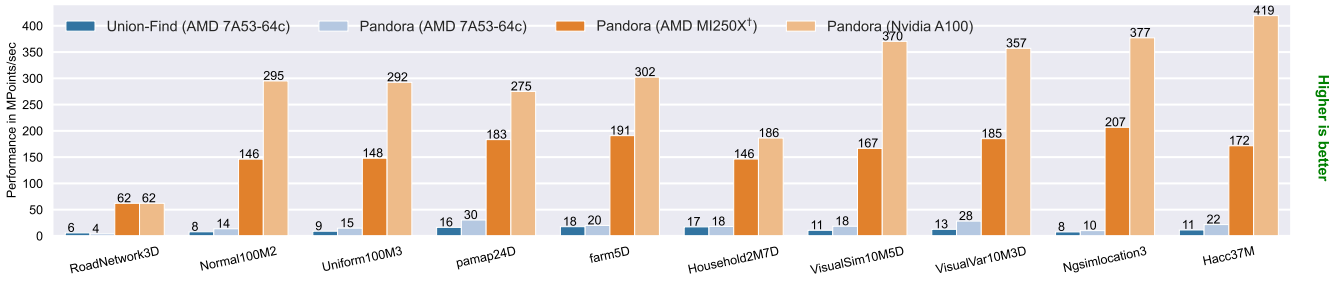
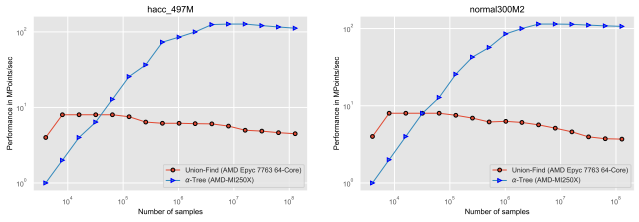


Figure 11: Performance comparison of the multithreaded (using AMD EPYC 7A53) and parallel (using Nvidia A100 and AMD MI250X (single GCD)) implementations.



(a) Hacc497M **(b) Normal300M2**

Figure 12: Effect of the dataset size on the parallel performance using AMD MI250X.

5.1.2 Testing environment. The numerical studies presented in the paper were performed using AMD EPYC 7A53 (64 cores), Nvidia A100 and a single GCD (Graphics Compute Die) of AMD MI250X[1]. The chips are based on TSMC's N7+, N7 and N6 technology, respectively, and can be considered to belong to the same generation. We used Clang 14.0.0 compiler for AMD EPYC 7A53, NVCC 11.5 for Nvidia A100, and ROCm 5.4 for AMD MI250X.

Performance Metrics: We measure our performance in MPoints/sec, which represents the number of points (in millions) processed per second. This metric is computed as 1e-6 * #points in dataset/ Time to compute dendrogram.

5.1.3 Datasets. For our experiments, we used a combination of artificial and real-world datasets listed in Table 2 to comprehensively evaluate our algorithm and meet our study goals. The GPS locations and HACC datasets replicate real-world conditions, the datasets generated with [15] allow for better comparison with other works, and synthetic datasets help us understand our algorithm's behaviour in different scenarios. We focused on 2D and 3D datasets, where dendrogram construction was the bottleneck; on higher dimensions, MST construction was the primary issue.

Table 2 also includes the information about the height of the constructed dendrograms. Specifically, we show the ratio of a dendrogram height for each dataset to that of a perfectly balanced binary tree. As we can see, it clearly indicates the skewedness of the dendrograms, highlighting the challenge of finding enough parallelization for an efficient algorithm.

[1]Currently, HIP (Heterogeneous-computing Interface for Portability) – the programming interface provided by AMD – only allows the use of each GCD as an independent GPU.

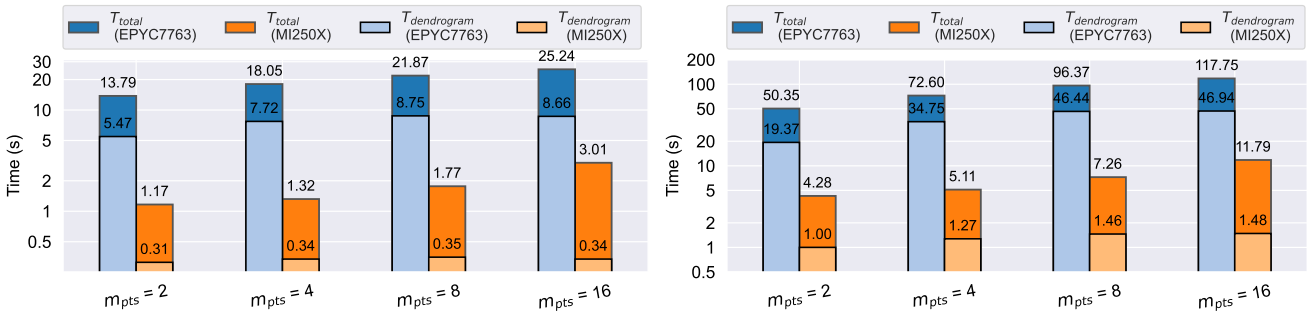
5.2 Performance evaluation of dendrogram construction

We evaluate the performance of the for the baseline UnionFind-MT and Pandora on various architecture across different datasets. The results are shown in Figure 11.

5.2.1 Multithreaded performance. We found that Pandora outperforms UnionFind-MT in multithreaded scenarios, with speed-ups ranging from 0.66 to 2.2 times. The *RoadNetwork3D* dataset is an exception, showing slower performance and having the smallest size with a lower dendrogram imbalance. Smaller 2D datasets exhibit limited multi-threading capabilities, with Pandora having a slight advantage. However, 3D and 4D datasets show more significant speed-ups, reaching up to 2.2 times faster. In higher-dimensional data sets, both algorithms have higher overall throughput, but Pandora still has a slight edge. Thus, even with twice the sequential work, Pandora remains faster than UnionFind-MT in a multithreaded setting.

5.2.2 GPU performance. Our study in Figure 11 showcases Pandora's performance on both Nvidia A100 and AMD MI250X. Our findings indicate that Pandora operates 6-20× faster on AMD MI250X (single GCD) than on AMD EPYC 7A53, whereas Nvidia A100 outperforms the fastest multithreaded variant by 10-37×. The *RoadNetwork3D* exhibits the lowest performance, caused by the small dataset size, not allowing to reach GPU saturation. We generally observe higher speed-ups for lower-dimensional datasets. This may only sometimes be the case as higher-dimensional datasets often have lower-dimensional substructures, resulting in similar behavior to lower-dimensional data. We also observe that the GPU variant performs well for all ranges of dendrogram skewness. For example, despite an imbalance of 43 in *VisualSim10M5D*, we still achieve a considerable speed-up over multithreaded Pandora. We conclude that Pandora has sufficient parallelism to utilize modern GPUs. , and it works well with highly skewed and not-so-skewed dendrograms alike.

Our implementation's performance is portable across multicore and GPU architectures, with a single source compiled for various backends. We did not optimize our algorithm specifically for any



(a) Hacc37M (b) Uniform100M3D

Figure 13: Comparison of the time to compute first two steps of the HDBSCAN* algorithm using MEMOGFK on AMD EPYC 7A53 (blue) and ArborX+our dendrogram algorithm using AMD MI250X (single GCD) (orange).

device architecture, nor did we investigate the impact of architectural differences between AMD MI250X and Nvidia A100 on performance. However, we should note that we primarily utilized Nvidia A100 in our software development, which may have led to a performance bias towards that architecture.

5.3 Scaling problem sizes

One way to determine the effectiveness of a parallel algorithm is to understand the smallest problem size at which it achieves peak performance. To this end, we studied the performance of the PANDORA algorithm with respect to the size of the dataset. We randomly sampled a large dataset to maintain a given distribution, as the algorithm could potentially be sensitive to the distribution of the data. The results of sampling three datasets (*Hacc497M*, *Normal300M2*, and *Uniform300M3*) on AMD MI250X are shown in Figure 12. For reference, we also show the performance of the UNIONFIND-MT implementation on AMD EPYC 7A53. For the UNIONFIND-MT implementation, performance immediately reaches its peak and slowly decreases afterward. On the other hand, the performance of the new algorithm increases with the number of samples until it reaches saturation and stays constant afterward. At around 30,000 samples, the performance of PANDORA-GPU exceeds that of UNIONFIND-MT. We can observe that GPU saturation occurs around the 10^6 mark, which is typical for GPU algorithms.

5.4 HDBSCAN* Performance

HDBSCAN* Parameters selection: The only parameter relevant to our evaluation is m_{pts}, which is the number of points to compute the core-distance. Different values of m_{pts} produce different dendrograms and affect the time spent on MST and dendrogram construction. We use the default $m_{pts} = 2$ in all our experiments except for Figure 13 where we evaluate the performance of HDBSCAN* for different values of m_{pts}. The performance gains of PANDORA over UNIONFIND-MT increases with m_{pts}, therefore, for a fair comparison we use $m_{pts} = 2$ in other experiments.

We evaluated the impact of PANDORA algorithm on HDBSCAN computation on AMD EPYC 7A53 and AMD MI250X. We used a multithreaded HDBSCAN* implementation MEMOGFK [43] as baseline. Additionally, we used GPU MST computation from ArborX [36] along with PANDORA for computing dendrogram for GPU implementation of HDBSCAN*. We based our results on two datasets: *Hacc37M* and *Uniform100M3D*, primarily focusing on the m_{pts} value, the sole parameter affecting these phases.

We show the results in Figure 13. Overall combination of ArborX with PANDORA on AMD MI250X is 8-12× faster than multithreaded MEMOGFK. And dendrogram computation with PANDORA on AMD MI250X is 17-33× faster than UNIONFIND-MT in MEMOGFK. We also observe that PANDORA uses less than a third of the total HDBSCAN* time, while UNIONFIND-MT can account for more than half.

With increasing m_{pts} times for dendrogram computation for both datasets. Going from $m_{pts} = 2$ to $m_{pts} = 16$, dendrogram computation time in PANDORA increased by 1.1-1.5×. In contrast, for UNIONFIND-MT, the this time increased by a factor of 1.6-2.4×

We also observe that the speed-up of dendrogram computation increases with m_{pts}. However, as m_{pts} rises, the EMST computation demands more resources. Thus, the benefits of quicker dendrogram computation may be counterbalanced by the more demanding EMST computation.

REFERENCES

[1] 2018. Next Generation Simulation (NGSIM) Vehicle Trajectories and Supporting Data. Available online: https://catalog.data.gov/dataset/next-generation-simulation-ngsim-vehicle-trajectories-and-supporting-data. Accessed: 2021-03-06.
[2] 2024. IKONOS Satellite Image of Tadco Farms, Saudi Arabia. https://www.satimagingcorp.com/gallery/ikonos/ikonos-tadco-farms-saudi-arabia. Accessed: 2024-01-01.
[3] Kevin Bache and Moshe Lichman. 2013. UCI machine learning repository. (2013).
[4] Bentley and Friedman. 1978. Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces. *IEEE Trans. Comput.* C-27, 2 (Feb. 1978), 97–105. https://doi.org/10.1109/TC.1978.1675043 Conference Name: IEEE Transactions on Computers.
[5] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Julian Shun. 2012. Internally deterministic parallel algorithms can be fast. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP '12)*. Association for Computing Machinery, New York, NY, USA, 181–192. https://doi.org/10.1145/2145816.2145840
[6] Nicolas Blin, Edwin Carlinet, Florian Lemaitre, Lionel Lacassagne, and Thierry Géraud. 2022. Max-Tree Computation on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3520–3531.
[7] Otakar Borůvka. 1926. O jistém problému minimálním. *Práce Mor. Prírodved. Spol. v Brne (Acta Societ. Scienc. Natur. Moravicae)* 3, 3 (1926), 37–58.

[8] Petra Bosilj, Ewa Kijak, and Sébastien Lefèvre. 2018. Partition and inclusion hierarchies of images: A comprehensive survey. *Journal of Imaging* 4, 2 (2018), 33.

[9] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. 2015. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data* 10, 1 (July 2015), 5:1–5:51. https://doi.org/10.1145/2733381

[10] The Thrust Developers. 2024. Thrust: A Parallel Algorithms Library. https://github.com/NVIDIA/thrust. Accessed: 2024-01-18.

[11] Dagmar Divjak, Nick Fieller, Dylan Glynn, and Justyna A Robinson. 2014. Finding structure in linguistic data. *Corpus methods for semantics: Quantitative studies in polysemy and synonymy* (2014), 405–441.

[12] James S Farris. 1972. Estimating phylogenetic trees from distance matrices. *The American Naturalist* 106, 951 (1972), 645–668.

[13] Eric Feigelson. 2012. Classification in Astronomy. *Advances in Machine Learning and Data Mining for Astronomy* (2012), 1.

[14] Marek Gagolewski, Maciej Bartoszuk, and Anna Cena. 2016. Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm. *Information Sciences* 363 (2016), 8–23.

[15] Junhao Gan and Yufei Tao. 2017. On the Hardness and Approximation of Euclidean DBSCAN. *ACM Transactions on Database Systems* 42, 3 (July 2017), 14:1–14:45. https://doi.org/10.1145/3083897

[16] John C Gower and Gavin JS Ross. 1969. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 18, 1 (1969), 54–64.

[17] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, et al. 2016. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42 (2016), 49–65.

[18] Michael Hahsler, Matthew Piekenbrock, and Derek Doran. 2019. dbscan: Fast Density-Based Clustering with R. *Journal of Statistical Software* 91, 1 (2019), 1–30. https://doi.org/10.18637/jss.v091.i01

[19] Jiří Havel, François Merciol, and Sébastien Lefèvre. 2013. Efficient schemes for computing α-tree representations. In *Mathematical Morphology and Its Applications to Signal and Image Processing: 11th International Symposium, ISMM 2013, Uppsala, Sweden, May 27-29, 2013. Proceedings 11*. Springer, 111–122.

[20] Jiří Havel, François Merciol, and Sébastien Lefèvre. 2019. Efficient tree construction for multiscale image representation and processing. *Journal of Real-Time Image Processing* 16 (2019), 1129–1146.

[21] Tony Hodge and M Jamie TV Cope. 2000. A myosin family tree. *Journal of cell science* 113, 19 (2000), 3353–3354.

[22] Todd Hricik, David Bader, and Oded Green. 2020. Using RAPIDS AI to accelerate graph data science workflows. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–4.

[23] Prasanta K Jana and Azad Naik. 2009. An efficient minimum spanning tree based clustering algorithm. In *2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*. IEEE, 1–5.

[24] Angur Mahmud Jarman. 2020. Hierarchical cluster analysis: Comparison of single linkage, complete linkage, average linkage and centroid linkage method. *Georgia Southern University* 29 (2020).

[25] Manohar Kaul, Bin Yang, and Christian S Jensen. 2013. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In *2013 IEEE 14th International Conference on Mobile Data Management*, Vol. 1. IEEE, 137–146.

[26] Joseph B. Kruskal. 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proc. Amer. Math. Soc.* 7, 1 (1956), 48–50. https://doi.org/10.2307/2033241 Publisher: American Mathematical Society.

[27] Michael Laszlo and Sumitra Mukherjee. 2005. Minimum spanning tree partitioning algorithm for microaggregation. *IEEE Transactions on Knowledge and Data Engineering* 17, 7 (2005), 902–911.

[28] Ivica Letunic and Peer Bork. 2007. Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display and annotation. *Bioinformatics* 23, 1 (2007), 127–128.

[29] Yong H Li and Anil K Jain. 1998. Classification of text documents. *Comput. J.* 41, 8 (1998), 537–546.

[30] Wei Luo, Alan M MacEachren, Peifeng Yin, and Frank Hardisty. 2011. Spatial-social network visualization for exploratory data analysis. In *Proceedings of the 3rd ACM SIGSPATIAL international workshop on location-based social networks*. 65–68.

[31] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software* 2, 11 (March 2017), 205. https://doi.org/10.21105/joss.00205

[32] Katie Ovens, B Frank Eames, and Ian McQuillan. 2021. Comparative analyses of gene co-expression networks: Implementations and applications in the study of evolution. *Frontiers in Genetics* 12 (2021), 695399.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[34] SP Pfeifer. 2020. The Molecular Evolutionary Clock. Theory and Practice. (2020).

[35] R. C. Prim. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36, 6 (Nov. 1957), 1389–1401. https://doi.org/10.1002/j.1538-7305.1957.tb01515.x Conference Name: The Bell System Technical Journal.

[36] Andrey Prokopenko, Piyush Sao, and Damien Lebrun-Grandie. 2023. A single-tree algorithm to compute the Euclidean minimum spanning tree on GPUs. In *Proceedings of the 51st International Conference on Parallel Processing (ICPP '22)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3545008.3546185

[37] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*. IEEE, 108–109.

[38] Robin Sibson. 1973. SLINK: an optimally efficient algorithm for the single-link cluster method. *The computer journal* 16, 1 (1973), 30–34.

[39] Pierre Soille. 2008. Constrained connectivity for hierarchical image partitioning and simplification. *IEEE transactions on pattern analysis and machine intelligence* 30, 7 (2008), 1132–1145.

[40] Robert E Tarjan and Jan Van Leeuwen. 1984. Worst-case analysis of set union algorithms. *Journal of the ACM (JACM)* 31, 2 (1984), 245–281.

[41] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. 2022. Kokkos 3: programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (April 2022), 805–817. https://doi.org/10.1109/TPDS.2021.3097283 Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[42] Manik Varma and Andrew Zisserman. 2003. Texture classification: Are filter banks necessary?. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, Vol. 2. IEEE, II–691.

[43] Yiqiu Wang, Shangdi Yu, Yan Gu, and Julian Shun. 2021. Fast parallel algorithms for Euclidean minimum spanning tree and hierarchical spatial clustering. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD/PODS '21)*. Association for Computing Machinery, 1982–1995. https://doi.org/10.1145/3448016.3457296

[44] Joe H Ward Jr. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association* 58, 301 (1963), 236–244.

[45] Ying Xu, Victor Olman, and Edward C Uberbacher. 1996. A segmentation algorithm for noisy images. In *Proceedings IEEE International Joint Symposia on Intelligence and Systems*. IEEE, 220–226.

[46] Ying Xu, Victor Olman, and Dong Xu. 2002. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics* 18, 4 (2002), 536–545.

[47] Odilia Yim and Kylee T Ramdeen. 2015. Hierarchical cluster analysis: comparison of three linkage measures and application to psychological data. *The quantitative methods for psychology* 11, 1 (2015), 8–21.

[48] Charles T Zahn. 1971. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on computers* 100, 1 (1971), 68–86.

[49] Emile Zuckerkandl and Linus Pauling. 1965. Evolutionary divergence and convergence in proteins. In *Evolving genes and proteins*. Elsevier, 97–166.