RESEARCH-ARTICLE

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

PIYUSH SAO, Oak Ridge National Laboratory, Oak Ridge, TN, United States

ANDREY PROKOPENKO, Oak Ridge National Laboratory, Oak Ridge, TN, United States

DAMIEN LEBRUN-GRANDIE, Oak Ridge National Laboratory, Oak Ridge, TN, United States

Open Access Support provided by:

Oak Ridge National Laboratory

Published: 12 August 2024

Citation in BibTeX format

ICPP '24: the 53rd International Conference on Parallel Processing
August 12 - 15, 2024
Gotland, Sweden

https://doi.org/10.1145/3673038.3673148
ISBN: 9798400717932

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

Piyush Sao*
saopk@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Andrey Prokopenko
prokopenkoav@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Damien Lebrun-Grandie
lebrungrandt@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

ABSTRACT

This paper introduces Pandora, a parallel algorithm for computing dendrograms, the hierarchical cluster trees for single linkage clustering (SLC). Current parallel approaches construct dendrograms by partitioning a minimum spanning tree and removing edges. However, they struggle with skewed, hard-to-parallelize real-world dendrograms. Consequently, computing dendrograms is the sequential bottleneck in HDBSCAN*[21], a popular SLC variant.

Pandora uses recursive tree contraction to address this limitation. Pandora contracts nodes to construct progressively smaller trees. It computes the smallest contracted dendrogram and expands it by inserting contracted edges. This recursive strategy is highly parallel, skew-independent, work-optimal, and well-suited for GPUs and multicores.

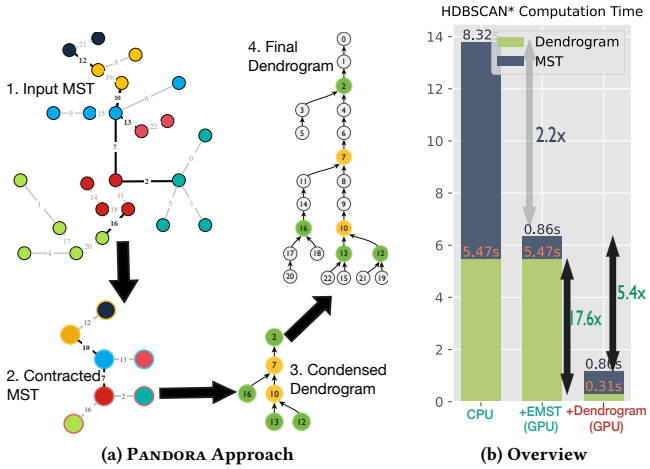
We develop a performance portable implementation of Pandora in Kokkos[31] and evaluate its performance on multicore CPUs and multi-vendor GPUs (e.g., Nvidia, AMD) for dendrogram construction in HDBSCAN*. Multithreaded Pandora is 2.2x faster than the current best-multithreaded implementation. Our GPU version achieves 6-20x speedup on AMD GPUs and 10-37x on NVIDIA GPUs over multithreaded Pandora. Pandora removes HDBSCAN*'s sequential bottleneck, greatly boosting efficiency, particularly with GPUs.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; • **General and reference** → *Performance*; • **Theory of computation** → *Sparsification and spanners*; *Computational geometry*.

ACM Reference Format:
Piyush Sao, Andrey Prokopenko, and Damien Lebrun-Grandie. 2024. PANDORA: A Parallel Dendrogram Construction Algorithm for Single Linkage Clustering on GPU. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3673038.3673148

ICPP '24, August 12–15, 2024, Gotland, Sweden
2024. ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673148



(a) Pandora Approach (b) Overview

Figure 1: Overview of proposed algorithm and results. Fig. 1a illustrates the Pandora algorithm using recursive tree contraction. The top left shows the original minimum spanning tree (MST); the bottom left shows the contracted MST. The dendrogram corresponding to the contraction is presented in the bottom right. Reinserting the contracted edges reconstructs the complete dendrogram. Note: The dendrogram only shows internal nodes corresponding to MST edges; leaf nodes for data points are omitted. Fig. 1b shows 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.

1 INTRODUCTION

Single-Linkage Clustering (SLC), a widely used hierarchical method, is applied in diverse fields such as bioinformatics [28], astronomy [9], and image analysis [6, 14, 29, 34]. SLC computation typically involves constructing a minimum spanning tree (MST) and computing a dendrogram to represent hierarchical clusters from the MST. MST computation may differ between SLC variants and data types (graphs, spatial points, or images); dendrogram computation is common among all variants. This paper focuses on parallel dendrogram computation arising from SLC computations.

Computing the dendrogram is especially challenging in the popular SLC variant, *Hierarchical Density-Based Spatial Clustering of Applications with Noise* (Hdbscan*) [7]. Hdbscan* hierarchically clusters spatial data by: 1. building a Euclidean minimum spanning

tree (EMST) using the mutual reachability distance and **2.** converting the EMST into a dendrogram [7]. Although parallel EMST construction is well-studied [25, 33], efficient parallel dendrogram computation remains challenging. Sequential or coarsely multi-threaded dendrogram construction often dominates runtime (sometimes exceeding 90% [33]), limiting HDBSCAN* performance on large spatial data. For example, Figure 1b (middle) shows that for the *Hacc37M* astronomy dataset, dendrogram construction on CPU consumes 86% of the overall time compared to MST construction on GPU. *We introduce a novel parallel GPU algorithm for dendrogram construction to address this bottleneck.*

Given the MST, there are two approaches to compute the dendrogram: **1.** *Top-down* [7, 33]: The top-down method repeatedly removes edges from the MST, splitting it into smaller subtrees representing clusters, until all edges are removed. However, it performs poorly on highly skewed dendrograms with heights exceeding $O(\log n)$, where n is the number of points. Such skewed dendrograms limit parallelism, resulting in quadratic work complexity and slow performance [33]. **2.** *Bottom-up*: The bottom-up approach starts with the smallest MST edge and iteratively adds edges to expand the cluster until all points are included. This approach is asymptotically work-optimal regardless of dendrogram skewness (Figure 1b, middle uses the bottom-up approach). However, it requires sequential edge addition for correctness, making it highly sequential. The bottom-up approach is currently the fastest method, but developing an asymptotically work-optimal parallel algorithm for dendrogram construction remains an open challenge.

We introduce PANDORA, a novel parallel algorithm for constructing dendrograms. PANDORA uses parallel-tree contraction to contract edges in the minimum spanning tree (MST), producing a reduced MST. We efficiently compute the dendrogram for the reduced MST and reintegrate the contracted edges. We recursively apply this process to construct the complete dendrogram asymptotically optimally. Figure 1a illustrates PANDORA using an MST with one level of contraction.

To implement this strategy, we must solve two problems: **1.** *Identifying contractible edges.* We aim to contract the maximum edges, but not all contractions are valid. We have characterized valid edge contractions and identified the maximal set that ensures the correctness of the final dendrogram. **2.** *Efficiently assembling the final dendrogram from its contracted version.* Inserting each contracted edge into the condensed dendrogram is highly parallel, but the naive approach is computationally costly and not asymptotically work-optimal. We leverage the algorithm's recursive nature and all levels of the condensed dendrogram to insert each contracted edge efficiently.

PANDORA consists of three main steps: **a)** finding edges to contract, **b)** contracting the minimum spanning tree (MST), and **c)** rebuilding the original dendrogram. These steps are highly parallelizable and can be easily adapted for multicore CPUs and GPU architectures using parallel constructs such as parallel-for, reduce, and scan. We implemented our algorithm using the performance-portable Kokkos library [31], marking the first known GPU implementation for dendrogram construction. We evaluated our algorithm on real-world and artificial datasets, comparing it to the best-known open-source multi-core CPU implementation. Our multi-core CPU implementation was twice as fast as the state-of-the-art, while the GPU variant achieved a 15-40× speedup compared to the multi-core CPU performance. Figure 1b (right) shows that we achieved our goal of improving the construction time by 17×, reducing it to 26% of the overall time for this dataset.

In summary, this paper makes the following contributions:

- We propose PANDORA a novel parallel dendrogram construction algorithm that overcomes the limitations of traditional top-down and bottom-up methods.
- We prove that any algorithm for computing a dendrogram requires $\Omega(n \log n)$ work. Our algorithm achieves this lower bound and is thus the *first work-optimal parallel algorithm* for dendrogram construction. In contrast, the best previously known parallel algorithm[33] only achieves this bound in the *expected* sense, and not in the worst case[1].
- We comprehensively analyze the edge contraction technique, establishing the necessary and sufficient conditions for the correctness of any algorithm based on tree-contraction[27].
- We present the first known GPU implementation for dendrogram construction, along with performance evaluations showing significant speedups compared to state-of-the-art multi-core CPU implementations.

Our work significantly advances the state-of-the-art in parallel HDBSCAN* clustering computation on GPUs for large datasets. When combined with recent developments in parallel Euclidean MST computation on GPUs [25], our algorithm enables rapid HDBSCAN* clustering on modern hardware. For example, a single Nvidia A100 GPU can compute HDBSCAN* clustering in under one second for a 37M cosmological dataset and around six seconds for a 300M uniformly distributed point cloud. This paper focuses on HDBSCAN*, yet PANDORA applies to any single-linkage clustering algorithm.

An extended version of the paper is available in Arxiv[27] and the performance portable implementation of HDBSCAN* using Euclidean MST computation from [25] and PANDORA is available with Arborx[20]:https://github.com/arborx/ArborX.

2 BACKGROUND

2.1 Hierarchical Single Linkage Clustering

2.1.1 Hierarchical Clustering. Hierarchical clustering generates a dendrogram, a cluster hierarchy, without specifying the number of clusters upfront. It has two variants: agglomerative (bottom-up) and divisive (top-down). The distance metric determines data point similarity, resulting in single, complete, and average linkage variations.

2.1.2 Single Linkage Clustering. Single linkage clustering (SLC) [11] defines the distance between clusters as the minimum distance between any two member points. Unlike complete and average linkage, SLC excels at detecting irregularly shaped clusters. Applications include bioinformatics [28], astronomy [9], image processing [6, 14, 29, 34], and others [16, 19, 35, 36]. SLC constructs a minimum spanning tree (MST) from a distance matrix or spatial search trees like kd-trees for spatial points [4, 25, 33], using Prim's, Kruskal's, or Borůvka's algorithm [5, 18, 24]. The MST then forms a dendrogram revealing the hierarchical cluster structure.

[1]To our knowledge, the worst-case runtime upper bound for[33] is $O(n \log^2 n)$

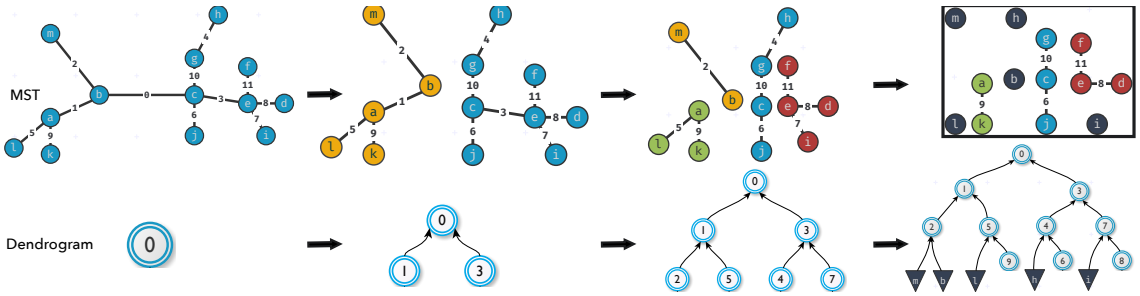


Figure 2: Top-down dendrogram construction steps (Section 2.2.2). The dendrogram root is the MST's heaviest edge, which divides the tree into two components when removed. Subsequently, each component's heaviest edge is identified, and its parent is the previous step's heaviest edge. This repeats recursively for each tree component.

Table 1: Available open-source dendrogram construction implementations

Implementation	Description
`scikit-learn` (Python) [23]	Sequential implementation
`hdbscan` (Python) [21]	Sequential implementation
`hdbscan` (R) [13]	Sequential implementation
Wang *et al.* [33]	Multi-threaded implementation in shared memory
`rapidsai` [15]	Parallel MST implementation using GPUs, sequential dendrogram construction

Algorithm 1 Top-down dendrogram algorithm[7]

```
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
```

2.1.3 HDBSCAN.* Hdbscan* (hierarchical DBSCAN) [7] is a hierarchical density-based clustering algorithm that groups points by local density using *minPts*. It adapts Euclidean distance into the density-aware *mutual reachability distance*. By using a dendrogram with clusterings at various ϵ, Hdbscan* avoids manually setting a distance threshold. Table 1 lists current Hdbscan* implementations. Prior work parallelized Hdbscan* by optimizing distance and MST computations, including RAPIDS.ai/Raft's single-linkage clustering [15]. To our knowledge, only Wang et al. [33] parallelized Hdbscan* dendrogram construction on multithreaded systems.

2.2 Dendrogram and construction algorithms

2.2.1 Dendrogram and its structure. **Dendrogram:** A dendrogram represents hierarchical clustering using a directed tree structure. Data points are leaf nodes, while clusters are internal nodes. Directed edges show the containment relationships between clusters. **Dendrogram in Single-Linkage Clustering:** In single-linkage clustering, the dendrogram's internal nodes are represented by minimum spanning tree (MST) edges. Clusters are defined by MST edges; removing an edge splits a cluster into two. Since removing an edge can only divide a cluster into two parts, the resulting dendrogram is usually a binary tree.

2.2.2 ***Top-down dendrogram construction.*** The top-down dendrogram construction algorithm uses divide-and-conquer to repeatedly split the graph by removing the heaviest edges[7].

The algorithm works as follows: **1.** Remove the heaviest edge in the MST. **2.** Split the tree into two subtrees. **3.** Recursively construct dendrograms for each subtree under the removed edge's parent node. **4.** Repeat steps 1-3 for all edges. Figure 2 illustrates the first three steps of this algorithm for an example MST, and Algorithm 1 presents the corresponding pseudo-code.

Limitations of the Top-Down Approach: The top-down dendrogram construction often struggles with real-world data's skewed dendrograms. Removing the heaviest edge frequently splits the tree into one large and one single-vertex component, instead of balanced components. **1.** *Increased Asymptotic Cost:* For highly skewed dendrograms, the algorithm's $O(nh)$ cost can exceed $O(n \log n)$ for balanced dendrograms, lacking work-optimality. **2.** *Limited Parallelism:* The dendrogram's skew limits parallelism. After removing several edges, the graph typically divides into a few large components and many isolated vertices or minor components. This imbalance and the $O(h)$ computation depth, potentially much higher than $O(\log n)$, restrict parallel processing.

2.2.3 Parallel variants of the top-down approach. Wang et al.[33] introduced a parallel variant of the top-down dendrogram construction algorithm to address the limitations of the sequential version, where removing a single edge often results in highly skewed partitions. The parallel variant simultaneously removes the top $n/2$ edges, splitting the graph into $n/2 + 1$ components. Dendrograms are then independently computed for each component before being stitched together to form the final dendrogram. They implemented

the algorithm in a multithreaded setting and showed that it offered more parallelism than its sequential counterpart.

Limitations: The parallel variant offers greater parallelism but shares similar shortcomings with the sequential version: **1.** Skewed partitions lead to work inefficiency, with the multithreaded evaluation in [33] showing dendrogram construction around 10× slower in some cases. **2.** Heavy reliance on parallel list-ranking, which significantly underperforms alternatives like prefix-sum or parallel sort on GPUs. **3.** Uneven parallelism and recursive nature pose additional challenges for the massively parallel GPU architecture.

Algorithm 2 Bottom-up dendrogram construction using union-find for a given minimum spanning tree $T = (V, E)$.

Require: MST $T = (V, E)$.
1: Sort E in descending order by weight, yielding $E_s = \{e_i\}$.
2: Initialize:
- Empty union-find structure UF
- Output array 'parent' of size $2|E| + 1$
3: Notation: c_w = root edge of dendrogram containing vertex w.
4: **for** $i = |E| - 1$ to 0 **do**
5: $(u, v) \leftarrow E_s[i]$ ▷ Extract vertices of edge i
6: $c_u,\ c_v \leftarrow$ UF.find(u), UF.find(v) ▷ Find roots for u, v
7: **if** u **not found then**
8: add $\{u\}$ to UF
9: $c_u \leftarrow u$
10: **if** v **not found then**
11: add $\{v\}$ to UF
12: $c_v \leftarrow v$
13: UF.union(c_u, c_v) ▷ Merge components
14: parent[c_u] = parent[c_v] = $E_s[i]$ ▷ Set parent edges
Ensure: Dendrogram represented by 'parent' array mapping edges to parent edges.

2.2.4 Bottom-up dendrogram construction. The bottom-up algorithm constructs the dendrogram bottom-up, starting with the smallest edges and iteratively merging clusters. First, the algorithm sorts the Minimum Spanning Tree edges in descending order and initializes an empty Union-Find structure and an output 'parent' array. For each edge, it identifies the components containing vertices u and v in the Union-Find structure [30], merging existing components or adding new ones for missing vertices. The parent of the newly merged components is updated to the current edge. Algorithm 2 has a worst-case time complexity of $O(n \log n)$ due to the $O(n \log n)$ sorting and the $O(n.\mathcal{A}(n))$ dendrogram construction, where $\mathcal{A}(n)$ is the inverse Ackerman function [30]. However, the sequential loop limits parallel efficiency, motivating a search for a work-optimal, parallel algorithm suitable for GPUs.

3 PARALLEL DENDROGRAM COMPUTATION

3.1 High-level idea behind PANDORA

PANDORA builds upon two key concepts: *dendrogram chains* and recursive tree contractions.

Definition 1 (Dendrogram Chains)**.** A dendrogram chain consists of MST edges $\{e_i, e_j, e_k \ldots, e_l\}$ in parent-child relationships where

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

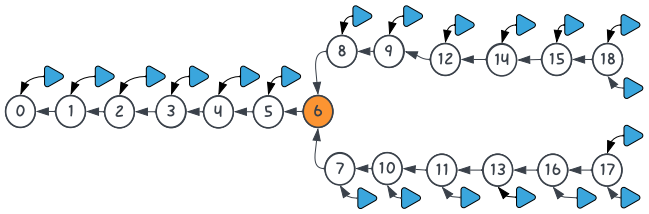


Figure 3: The PANDORA leverages dendrogram chains to construct them efficiently. This dendrogram can be divided into three chains: top, bottom-left, and bottom-right.

$p(e_i) = e_j$, $p(e_j) = e_k$, etc., with e_l having a parent outside the chain. e_l is the root of the chain.

For example, consider an inverted Y-shaped dendrogram (Fig. 3). It consists of three dendrogram chains—top, bottom-left, and bottom-right.

To build the complete dendrogram, PANDORA follows a three-step process: **1.** Assign each edge to a dendrogram chain using a tree contraction method. This generates a condensed version of the original dendrogram by condensing each chain into a single edge, allowing us to map all edges to their respective chains. **2.** Sort the edges within each chain by weight to form dendrogram chains. **3.** Connect the dendrogram chains to reconstruct the full dendrogram. We begin by defining essential terms and notations for our algorithm's description.

3.2 Terminology and notation

3.2.1 Minimum spanning tree structure. Consider a Minimum Spanning Tree (MST) $T = \{V, E, W\}$ for which we want to calculate its dendrogram. Let n_v be the number of vertices and $n = n_v - 1$ be

the number of edges in the MST. Computing the dendrogram begins by sorting the edges in T by descending weight, which takes $O(n \log n)$ time. This sorting step ensures the dendrogram's uniqueness and enables validating our method by consistently ordering edges with equal weights. For simplicity, we assume the edges are already sorted in the following discussions. We use this notation to describe the tree's incidence structure:

Incident edges Incident(v): The set of all edges incident to vertex $v \in V$. For example, in Figure 5a, Incident$(a) = \{e_0, e_2, e_3, e_5\}$.

Maximum incident edge maxIncident(v): The edge with the highest index in Incident(v). In the example, maxIncident$(a) = e_5$.

Neighboring edges $\mathcal{N}(e)$: For an edge e connecting vertices v and u, the set of edges that share a vertex with e, i.e., $\mathcal{N}(e) =$ Incident$(v) \cup$ Incident(u).

Edge contraction of a tree: To create a contracted tree $T_c = (V_c, E_c)$ from a tree T and an edge subset E_c, we contract the edges in $E - E_c$ as follows. Initialize $V_c = V$. For each edge $e = (u, v) \in E - E_c$: **1.** Merge u and v into a supervertex vu **2.** Remove u and v from V_c **3.** Add supervertex vu to V_c **4.** vu inherits the neighbors of u and v, excluding u and v. **5.** Repeat for all edges in $E - E_c$. The resulting T_c has the modified vertex set V_c and edge subset E_c.

3.2.2 Dendrogram structure. A dendrogram $\mathcal{D} = \{V_d, E_d\}$ is a directed rooted binary tree. The vertex set V_d contains two node types: vertex nodes (leaf nodes) representing individual data points, and edge nodes (internal nodes) representing clusters. The parent function P maps each node v to its parent u, defining directed links that establish parent-child relationships and the edge set E_d, which contains directed edges $(v \rightarrow u)$ for each node v, where $u = P(v)$. *Computing the dendrogram is equivalent to finding the parent P of each node.*

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, which sequentially removes edges in Incident(v) from heaviest to lightest. Thus, the dendrogram parent of vertex v is the edge in Incident(v) with the largest index:

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

For instance, in Figure 5a, $P(a) = e_5$. The tree's incidence structure allows determining the parents of all vertex nodes $v \in V$. However, the main challenge lies in *identifying the parents of the edge nodes.*

3.2.3 Types of edge nodes. Edge nodes in a binary dendrogram are classified by the number and type of their children. Each edge node has two children, which can be either edge or vertex nodes, leading to three types of edge nodes (Figure 4): **a) Leaf edges:** Edge nodes with two vertex node children. **b) Chain edges:** Edge nodes with one vertex node child and one edge node child. **c) α-Edges:** Edge nodes with two edge node children and no vertex node children. The MST's local incidence structure and Equation 1 allow classifying an edge node as a leaf, chain, or α-edge based on its children. However, identifying an edge node's parent remains challenging, as parent-child relationships often extend beyond neighboring nodes. For example, in Figure 5b, edge node e_2's parent is e_1, despite e_1 and e_2 being in separate tree sections.

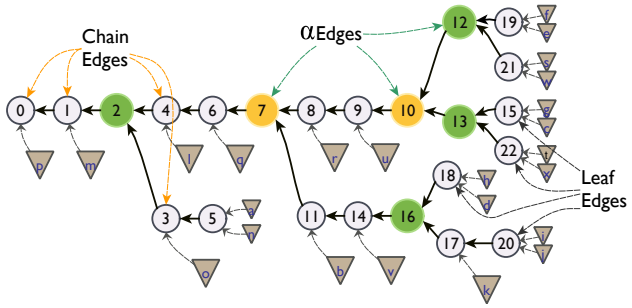


Figure 4: Different edge nodes in a dendrogram. We show the dendrogram corresponding to the MST in Figure 5a. We mark the vertex nodes as triangles and edge nodes as circles. The edge nodes are further classified into three types: leaf edges (with two vertex nodes as children), chain edges (with one vertex and one edge node as children), and α-edges (with two edge nodes as children).

3.2.4 Dendrogram chains and skewness. **Dendrogram chains:** A dendrogram chain is an unbranched lineage consisting of a series of chain edges that end in either a leaf or an α edge. Each chain edge node, except the last, has a single child: the next edge in the chain. Leaf chains are chains that end in a leaf edge.

Skewness of the Dendrogram: A dendrogram's skewness, defined as the ratio of its height to the ideal height of $\log_2 n$, can increase significantly due to a large number of chains. High skewness is commonly observed in real-world dendrograms, even in low-dimensional Gaussian dendrograms, posing challenges for developing parallel dendrogram algorithms. This skewness is prevalent across various datasets, from GPS locations to cosmology and power usage (see Table 2.)

3.3 Recursive tree contraction

Pandora condenses the minimum spanning tree (MST) by contracting all but the α edges, creating a simplified version. This condensed MST's dendrogram matches the full dendrogram's structure obtained by merging chain and leaf nodes. The simplified dendrogram effectively captures the essence of the full dendrogram.

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

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

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

Figure 5 demonstrates this process. In the Minimum Spanning Tree (MST) example shown in Figure 5a, the α-edges are highlighted in Figure 5b. For example, $e_{16} = \{i, d\}$ is an α-edge because maxIncident$(i) = 20$ and maxIncident$(d) = 18$, both different from 16. None of the terminal edges are α-edges. For instance, $e_1 = \{m, k\}$

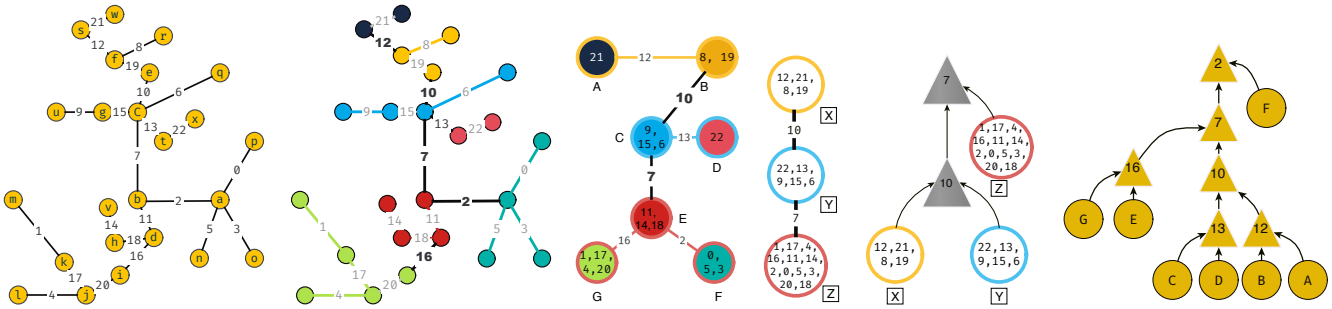


Figure 5: Fig. 5a shows the original MST. Fig. 5b highlights the α-edges, whose removal divides the tree into color-coded components. These components contract into α-vertices. Fig. 5c depicts the resulting α-MST, the first contraction level formed by α-vertices and α-edges. A second contraction level forms the β-MST (Fig. 5d). Both α-MST and β-MST show the contracted edges within supervertices. Finally, the α-dendrogram and β-dendrogram correspond to the α-MST (Fig. 5c) and β-MST (Fig. 5d) respectively.

is not an α-edge because maxIncident$(m) = 1$. Additionally, several internal edges like e_{20} and e_{17} are also non-α edges.

3.3.2 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 the 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 maintain a mapping between the original vertices and their T_α counterparts, which is crucial for tracing back to the original structure.

For example, Figure 5b shows an MST with highlighted α edges. We contract the non-α edges to obtain the contracted tree in Figure 5c. Vertices in Figure 5b merged to form a supervertex share the same color. In Figure 5c, vertices a, n, o, and p are merged into a single cyan-colored supervertex.

Multilevel tree contraction: We can compute the dendrogram of T_α by recursively contracting edges. In each iteration, we identify α-edges in the current contracted tree and contract the remaining edges to create the tree for the next iteration. The recursion stops when no α-edges remain, resulting in a single chain dendrogram obtained by sorting the edges by their indices.

Figure 5d shows the β-MST, which is the result of applying a second contraction level to the α-MST in Figure 5c. The β-MST cannot be further contracted, ending the recursion. Figure 5e represents the final contraction stage as the β-dendrogram.

In summary, we begin with a complete Minimum Spanning Tree (MST) and generate a series of smaller trees by applying multilevel tree contractions. This process iterates until the resulting tree contains no α edges. Sorting the edges of this final tree produces a highly compact single chain dendrogram. The following section describes how to expand this condensed dendrogram into a comprehensive representation.

3.4 Efficient Dendrogram Expansion

This section explains how to construct a complete dendrogram from a condensed dendrogram through a process called *expansion*. Section 3.4.1 describes single-level expansion, which is suboptimal

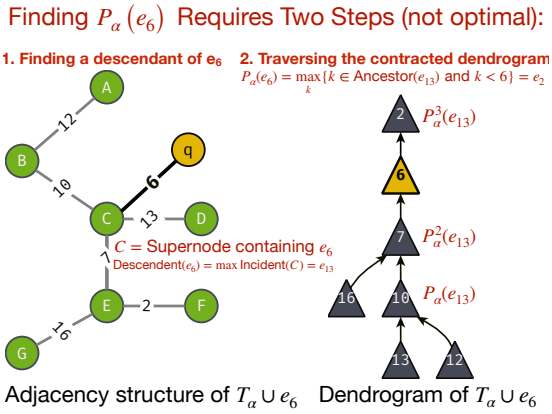
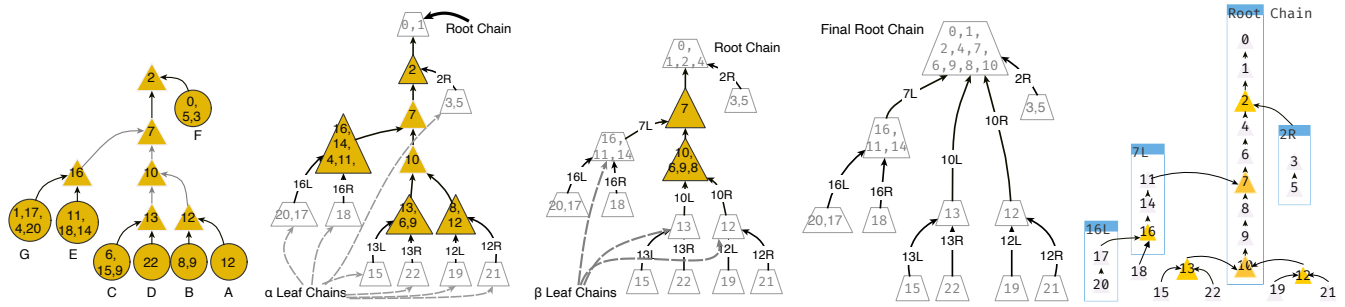


Figure 6: Inserting a non-α edge e_6 into the α-dendrogram. In this process, a single level contraction is done within the supervertex to find the parent of the edge. For example, in the case of edge e_6, we identify its parent by finding the maximum incident edge of the supervertex C, which is e_{13}. 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.

for reconstructing a dendrogram. To address this limitation, Section 3.4.2 presents an efficient expansion algorithm that utilizes all contraction levels.

3.4.1 Single-Level Dendrogram Expansion. Given an input Minimum Spanning Tree (MST) T, a contracted tree T_α containing all α edges, and the dendrogram of T_α specified by the parent-child relation P_α, the objective is to assign each non-α edge to a specific dendrogram chain. The assignment process follows these steps:

1. Find the α-vertex $V_\alpha(e)$ containing edge e. **2.** Determine $V_\alpha(e)$'s dendrogram parent $P_\alpha(V_\alpha(e))$ in the α-dendrogram. **3.** Traverse the α-dendrogram upwards from $P_\alpha(V_\alpha(e))$ until encountering an α edge with a smaller index than e. This edge becomes e's dendrogram parent, denoted as $P_\alpha(e)$.



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

Figure 7: Expanding the contracted dendrogram: 1. Figure 7a shows the α dendrogram, where triangles represent α edges and circles represent α vertices. The diagram displays non-α edges within their corresponding α vertices. 2. To identify non-α edges belonging to an α leaf chain, we compare each edge's index to its α parent's index (Figure 7a). Edges with indexes above the α parent belong to the α leaf chain. 3. For the remaining non-α edges, we determine if they belong to a β leaf chain by comparing each edge's index to the β edge, the parent of the previously identified α parent. Edges with indexes higher than the β parent are assigned to the β leaf chain (Figure 7c). This process continues until all edges are assigned to a leaf chain or no more contraction levels remain. 4. Any unassigned edges after the previous steps are allocated to the root chain if further contraction is not possible (Figure 7d). 5. Finally, the algorithm merges the sorted chains to form the final dendrogram (Figure 7e).

Consider mapping edge e_6 into the dendrogram for the minimum spanning tree (MST) shown in Figure 5. $V_\alpha(e_6) = C$ contains e_6 (Figure 5c). In the α-dendrogram, $V_\alpha(e_6)$'s parent is $P_\alpha(V_\alpha(e_6)) = e_{13}$ (Figure 5f). To find e_6's parent, traverse the alpha dendrogram bottom-up from e_{13}, seeking an α edge with a lower index than e_6 (Figure 6). The lower-indexed edge e_2 becomes e_6's α-parent. Since e_7 is placed on e_2's left side, e_6 is assigned to chain $2L$. However, this method is suboptimal as it requires bottom-up traversal of the alpha dendrogram for all non-α edges. In the worst case, the alpha dendrogram tree height can be $O(n)$, resulting in $O(n^2)$ work to find chains for all non-α edges.

3.4.2 Efficient Dendrogram Expansion from Multilevel Tree Contraction. Dendrogram expansion can be optimized with two key observations: First, edges in a leaf chain can be quickly identified without traversing the entire α dendrogram. Second, for edges not in an α dendrogram leaf chain, their membership in a β dendrogram leaf chain can be efficiently checked. By recursively applying this process, all edges are associated with a leaf chain at some level. Rather than inefficiently traversing the α dendrogram bottom-up, leaf chain membership is checked starting at the dendrogram level. This is more efficient since the number of contraction levels is $\log_2 n$.

Leaf Chains: Leaf chains are linked to their respective dendrograms. An α leaf chain is a sequence that concludes with a leaf edge in the dendrogram. For example, in Figure 7e, the sequence denoted by $16L$ qualifies as an α leaf chain. Removing all α leaf chains from a dendrogram reveals new β leaf chains ending with an α edge. 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: An edge's leaf chain membership at any level can be efficiently determined in constant time. The earliest level each edge joins a leaf chain is identified. For each non-α edge e, its membership in an α leaf chain is checked by comparing e's index to its α parent's index. If the α parent's index is lower, e is in an α leaf chain. Otherwise, e's membership in a β leaf chain is checked by comparing it to its β parent. Determining an edge's leaf chain membership at any level takes $O(1)$ time. Non-α edges are mapped to chains by checking their membership in α, β, etc. leaf chains until placed. Any remaining edges are grouped in the root chain. The cost of mapping an edge to a leaf chain is determined by the number of contraction levels.

Example: To map edge e_{15} to a chain: **a)** Its α-vertex is C (Figure 5c). **b)** $P_\alpha(C) = 13$ (Figure 5f). **c)** Since $15 > 13$, e_{15} is in the $13R$ leaf chain. For edge e_{11}: **a)** The α-vertex is E, with $P_\alpha(E) = 16$ (Figure 5f). **b)** As $16 > 11$, e_{11} is not in an α leaf chain. **c)** The β-vertex is X (Figure 5d), with $P_\beta(X) = 7$ (Figure 5e). **d)** Since $11 > 7$, e_{11} is in a β-leaf chain.

The mapping process: **1.** Start with the α-dendrogram (Figure 5f) **2.** Determine V_α for all edges (Figure 7a) **3.** Identify α leaf chain edges (Figure 7b) **4.** Check remaining edges for β leaf chains (Figure 7c) **5.** Assign non-leaf chain edges to the root chain (Figure 7d)

3.4.3 Final dendrogram construction. In the previous step, we assigned all edges to a leaf or root chain. Now, we use this to build the entire dendrogram: **a) Sorting Chains:** Sorting each chain forms partial dendrograms. In the sorted chain, we assign each edge's parent to its predecessor, except for the first edge, handled in the next step. **b) Stitching Chains**: Each chain is a leaf chain of a contracted edge from a contraction level. The parent of the first edge in a chain is the corresponding edge for that chain. For instance, the α-leaf chain $(17, 20)$ corresponds to α-edge e_{16} (Figure 7b), while β-leaf chain $(16, 11, 14)$ corresponds to β-edge e_7 (Figure 7c). Thus, the parent of e_{17} is α-edge e_{16}, and the parent of

e_{11} is β-edge e_7 (Figure 7e). Connecting chains this way forms the complete dendrogram (Figure 7e).

3.5 Performance portable implementation

Kokkos We used Kokkos [31], a performance-portable programming model for CPUs and GPUs from Nvidia, AMD, and Intel, to implement our algorithm. Kokkos abstracts execution ("execution space") and memory resources ("memory space"), but users must explicitly manage data movement between these spaces, as Kokkos does not perform hidden data copies. Kokkos provides parallel patterns, such as loops, reductions, and scans, that abstract hardware complexity.

Pandora implementation The Pandora algorithm has two main computational tasks: tree contraction and dendrogram expansion. During dendrogram expansion, Pandora maps contracted edges to their chains in parallel, avoiding significant load imbalance. Outside of Kokkos, parallel loops, reductions, prefix sums, and sorts used in these algorithms are widely available in many parallel libraries, such as Thrust [8].

3.6 Asymptotic Work Analysis

This section proves Pandora is work-optimal. We first establish the lower bound of $\Omega(n \log n)$ for any dendrogram computing algorithm and show that Pandora achieves this bound.

Asymptotic Lower Bounds for Dendrogram Computation Computing a dendrogram requires at least as many operations as sorting n floats. However, not all methods require sorting. For example, the top-down approach (Algorithm 1) can be performed without sorting. We prove that the lower bound for any dendrogram algorithm is $\Omega(n \log n)$.

Theorem-1: In the worst case, computing a dendrogram from a tree with n edges requires $\Omega(n \log n)$ operations.

Proof: To prove the lower bound, consider the problem of sorting n floats. First, construct a minimum spanning tree T with the floats as edge weights. The tree has $n+1$ vertices and n edges in a star topology, with one central vertex connected to all others. The dendrogram of T is a single chain of edges sorted by weight, allowing the floats to be extracted in sorted order in $O(n)$ time. If the dendrogram could be computed in *little-o* $o(n \log n)$ time, it would enable sorting the n floats faster than the $\Omega(n \log n)$ lower bound, which is impossible. Therefore, in the worst case, computing the dendrogram requires $\Omega(n \log n)$ operations.

Asymptotic Work Complexity for Pandora Pandora constructs the dendrogram of an MST with n edges in $O(n \log n)$ operations. We bound the number of edges in each level of the contracted MST, including leaf edges (n_l), chain edges (n_c), and α edges (n_α) (Section 3.2). We prove that $n_\alpha \leq (n-1)/2$ and the number of contraction levels is at most $\lceil \log_2(n+1) \rceil$. This implies the total cost of edge contractions is $O(n)$ and dendrogram expansion costs $O(n \log n)$. Sorting, done twice, costs $O(n \log n)$ each time. The overall cost is $O(n \log n)$, making the algorithm work-optimal.

Proof of $n_\alpha \leq (n-1)/2$: The number of leaf edges is always one more than the number of α-edges: $n_\alpha = n_l - 1$. Since the dendrogram has only α, leaf, and chain edges, $n_\alpha + n_l + n_c = n$. Substituting $n_\alpha = n_l - 1$, we have $n_c = n - 2n_\alpha - 1$. Chain edges are non-negative, i.e., $n_c \geq 0$, so:

$$n_\alpha \leq (n-1)/2.$$

Number of Contraction Levels: Each tree contraction halves the number of edges in the contracted MST. After k levels, the number of remaining edges is at most $(n - 2^k + 1)/2^k$. There are no edges left after k reaches $\lceil \log_2(n+1) \rceil$.

Cost of Edge Contraction: Tree contraction on a tree with n edges is equivalent to a prefix sum on an array with $2n$ entries [22], requiring $< c_1 n$ operations for some constant c_1. The total contraction cost is $< 2c_1 n$ operations.

Cost of Dendrogram Expansion: Mapping each edge to its dendrogram chain costs $O(\log n)$. Checking if an edge is in a leaf chain at the k-th contraction level is $O(1)$. With $O(\log n)$ contraction levels, mapping all n edges costs $O(n \log n)$.

Total Cost of Dendrogram Construction: Two $O(n \log n)$ sorting operations, along with $O(n)$ tree contraction and $O(n \log n)$ dendrogram expansion, make our algorithm work-optimal with an overall cost of $O(n \log n)$.

Table 2: Datasets used in experiments

Name	Dim	n_{pts}	Imb†	Ref.†	Desc.†
Ngsimlocation3	2	6M	1e3	[1]	GPS loc
RoadNetwork3	2	400K	150	[17]	Road network
Pamap2	4	3.8M	6e3	[26]	Activity monitoring
Farm	5	3.6M	5e4	[2]	VZ-features[32]
Household	7	2.0M	1e3	[3]	Household power
Hacc37M	3	37M	1e5	[12]	Cosmology
Hacc497M	3	497M	6e5	[12]	Cosmology
VisualVar10M2D	2	10M	3e3	[10]	GAN
VisualVar10M3D	3	10M	1e4	[10]	GAN
VisualSim10M5D	5	10M	43	[10]	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 EXPERIMENTAL RESULTS

In our implementation, we used ArborX [20] (version 1.4-devel) to compute EMST, and Kokkos [31] (version 3.7) for implementing our parallel dendrogram algorithm. The implemented algorithm is available in the main ArborX repository[2].

Testing environment. The numerical studies were performed using AMD EPYC 7A53 (64 cores), Nvidia A100 and a single GCD (Graphics Compute Die) of AMD MI250X[3]. The chips are based on TSMC's N7+, N7 and N6 technology and can be considered to belong to the same generation. We used Clang 14.0.0 compiler for

[2] https://github.com/arborx/ArborX

[3] Currently, HIP (Heterogeneous-computing Interface for Portability) – AMD's programming interface – only allows using each GCD as an independent GPU.

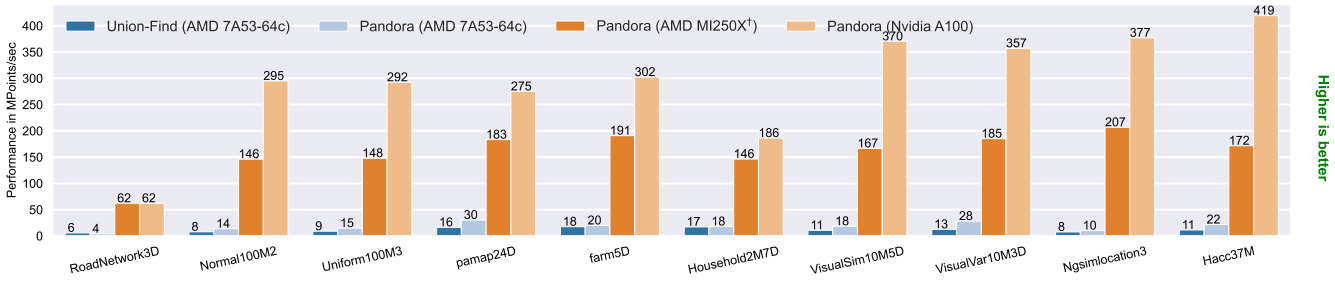
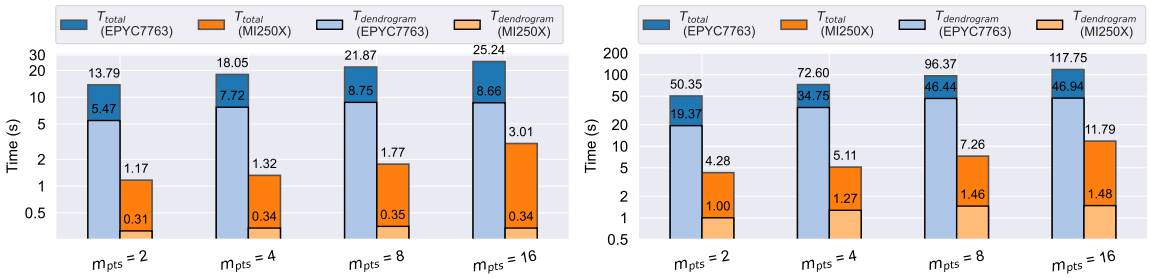
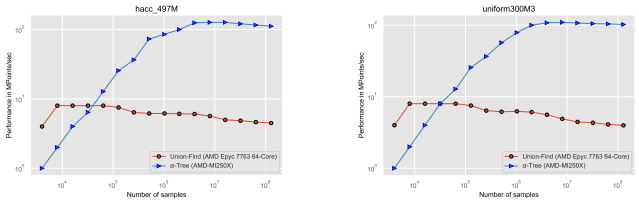


Figure 8: Performance comparison of the multithreaded (using AMD EPYC 7A53) and parallel (using Nvidia A100 and AMD MI250X (single GCD)) implementations.



(a) Hacc37M **(b) Uniform100M3D**

Figure 9: 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).



(a) Hacc497M **(b) Uniform300M3**

Figure 10: Effect of the dataset size on the parallel performance using AMD MI250X.

AMD EPYC 7A53, NVCC 11.5 for Nvidia A100, and ROCm 5.4 for AMD MI250X.

Datasets. We thoroughly evaluated our algorithm using the artificial and real-world datasets in Table 2. The GPS locations and HACC datasets mimic real-world conditions, while the datasets from [10] enable comparison to other works. Synthetic datasets elucidate our algorithm's behavior in different scenarios. For HDBSCAN* on low-dimensional datasets, dendrogram construction is the main bottleneck. However, as MST cost increases with dimensions, the dendrogram is not the bottleneck for higher-dimensional datasets. Therefore, we focus on datasets up to seven dimensions. Table 2 includes each dataset's dendrogram height ratio relative to a perfectly balanced binary tree, indicating the dendrograms' skewness and the difficulty of adequate parallelization for an efficient algorithm.

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 [33]. UNIONFIND-MT involves a parallel multithreaded sort and a sequential Union-find step. We used this implementation for verification. To our knowledge, this is the fastest available dendrogram computation implementation, and we use it as a baseline for comparison.

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

HDBSCAN* Parameters selection: The only relevant parameter is m_{pts}, the number of points to compute the core-distance. Different m_{pts} values produce different dendrograms, affecting the time spent on MST and dendrogram construction. We use the default $m_{pts} = 2$ except for Figure 9 where we evaluate HDBSCAN*'s performance for different m_{pts} values. PANDORA's performance gains over UNIONFIND-MT increase with m_{pts}, so for a fair comparison we use $m_{pts} = 2$ in other experiments.

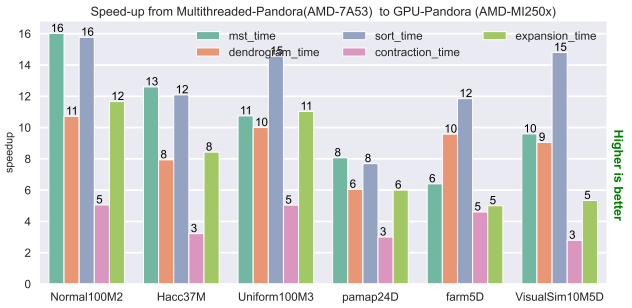


Figure 11: Speedup of AMD MI250X over AMD EPYC 7A53 for different phases of HDBSCAN* withPandora.

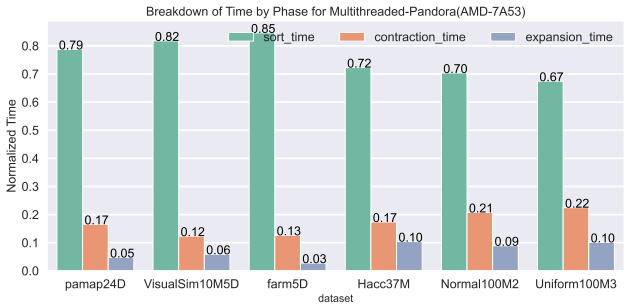


Figure 12: Breakdown of time spent Pandora in on AMD EPYC 7A53.

4.1 Performance of Dendrogram Construction

We evaluate the performance of the baseline UnionFind-MT and Pandora on various architectures across different datasets. The results are shown in Figure 8.

Multithreaded performance Pandora outperforms UnionFind-MT in multithreaded scenarios, with speed-ups of 0.66-2.2x, except for the *RoadNetwork3D* dataset which has the smallest size and lower dendrogram imbalance. Smaller 2D datasets have limited multi-threading scalability, while 3D and 4D datasets have more significant speed-ups, up to 2.2x faster. Pandora maintains a slight edge on higher-dimensional datasets, which have higher overall throughput. Despite twice the sequential work, Pandora remains faster than UnionFind-MT in a multithreaded setting.

GPU performance Pandora is 6-20× faster on AMD MI250X (single GCD) than on AMD EPYC 7A53, and Nvidia A100 outperforms the fastest multithreaded variant by 10-37×. The *RoadNetwork3D* has the lowest performance due to its small size not allowing GPU saturation. Lower-dimensional datasets generally have higher speed-ups, likely due to lower-dimensional substructures in higher-dimensional datasets. Pandora performs well for all ranges of dendrogram skewness, achieving considerable speed-ups over multithreaded Pandora even with an imbalance of 43 in *VisualSim10M5D*. Pandora has sufficient parallelism to utilize modern GPUs and works well with both highly skewed and not-so-skewed dendrograms.

Our implementation delivers portable performance across multicore and GPU architectures from a single source. We did not optimize for specific device architectures or investigate the impact of differences between AMD MI250X and Nvidia A100. However, primarily using Nvidia A100 in development may have led to a performance bias towards that architecture.

Scalability of different phases in Pandora Figure 11 shows the speed-up of AMD MI250X over AMD EPYC 7A53 for the following phases of Hdbscan* with Pandora: **a)** EMST construction **b)** Multilevel Contraction of MST **c)** Reconstruction of the dendrogram from the contracted MST **d)** Sorting (includes both initial and final sort, as well as other operations) Sorting is the most scalable phase, 10-20× faster than AMD EPYC 7A53, while multilevel edge contraction scales the worst at 3-5×. On AMD EPYC 7A53, sorting dominates the runtime, followed by multilevel edge contraction, with negligible dendrogram reconstruction time. Pandora achieves significant speed-ups on AMD MI250X compared to AMD EPYC 7A53, with strong overall performance despite multilevel edge contraction's scaling limitations.

Scaling Problem Size Determining the smallest problem size at which a parallel algorithm achieves peak performance helps gauge its effectiveness. We studied how dataset size affects the performance of the Pandora algorithm. We randomly sampled large datasets to test the algorithm's sensitivity to data distribution. Figure 10 shows the results for three datasets (*Hacc497M*, *Normal300M2*, and *Uniform300M3*) on AMD MI250X. For comparison, we also show UnionFind-MT's performance on AMD EPYC 7A53. UnionFind-MT's performance immediately peaks and then slowly decreases. In contrast, Pandora's performance increases with sample size until saturating. By 30,000 samples, Pandora-GPU outperforms UnionFind-MT. As typical for GPU algorithms, saturation occurs around 10^6 samples.

4.2 Hdbscan* Performance

We evaluated Pandora's impact on HDBSCAN computation on AMD EPYC 7A53 and AMD MI250X, using: **1.** MemoGFK [33], a multithreaded Hdbscan* implementation, as the baseline **2.** ArborX's [25] GPU MST computation and Pandora for the GPU Hdbscan* dendrogram computation Results were based on *Hacc37M* and *Uniform100M3D* datasets, focusing on the m_{pts} parameter, which solely affects these phases.

Figure 9 shows the results: **1.** ArborX with Pandora on AMD MI250X is 8-12× faster than multithreaded MemoGFK overall. Moreover, Pandora's dendrogram computation on AMD MI250X is 17-33× faster than MemoGFK's UnionFind-MT. **2.** Pandora uses less than a third of the total Hdbscan* time, whereas UnionFind-MT can exceed half. **3.** Increasing m_{pts} from 2 to 16 raised dendrogram computation time by 1.1-1.5× for Pandora and 1.6-2.4× for UnionFind-MT. **4.** The speed-up of dendrogram computation increases with m_{pts}. However, higher m_{pts} values demand more resources for EMST computation, potentially counterbalancing the benefits of quicker dendrogram computation.

5 CONCLUSION

We presented a new parallel algorithm to construct a dendrogram using GPUs and described the implementation details. Experimental

results confirmed the algorithm's performance, portability, and efficiency on various datasets and hardware architectures compared to existing approaches. To our knowledge, this is the first dendrogram construction algorithm on GPUs. Combining our algorithm with EMST on GPUs enables HDBSCAN* clustering in under a minute for GPU memory-fitting datasets; we plan to extend it to larger-than-memory datasets in the future. While this paper focuses on HDBSCAN*, PANDORA is applicable to any single-linkage clustering algorithm. Future work will focus on improving kernel efficiency and exploring alternate MST compression algorithms to leverage the untapped potential in this area.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

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] Otakar Borůvka. 1926. O jistém problému minimálním. *Práce Mor. Prírodved. Spol. v Brne (Acta Societ. Scienc. Natur. Moravicae)* 3, 3 (1926), 37–58.

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

[7] 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

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

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

[10] 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

[11] 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.

[12] 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.

[13] 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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, and S. R. Slattery. 2020. ArborX: A Performance Portable Geometric Search Library. *ACM Trans. Math. Softw.* 47, 1, Article 2 (Dec. 2020), 15 pages. https://doi.org/10.1145/3412558

[21] 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

[22] Gary L Miller and John H Reif. 1985. Parallel tree contraction and its application. In *FOCS*, Vol. 26. 478–489.

[23] 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.

[24] 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.

[25] 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

[26] 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.

[27] Piyush Sao, Andrey Prokopenko, and Damien Lebrun-Grandié. 2024. PANDORA: A Parallel Dendrogram Construction Algorithm for Single Linkage Clustering on GPU. *arXiv preprint arXiv:2401.06089* (2024).

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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

[34] 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.

[35] 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.

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